DEPARTAMENTO DE ARQUITECTURA Y TECNOLOGÍA DE SISTEMAS INFORMÁTICOS

Facultad de Informática
UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DOCTORAL

ARQUITECTURA MULTIAGENTE PARA E/S DE ALTO RENDIMIENTO EN CLUSTERS

Autora
María de los Santos Pérez Hernández
Licenciada en Informática

Directores
Jesús Carretero Pérez
Doctor en Informática

Félix García Carballeira
Doctor en Informática

Año: 2003

Tribunal

PRESIDENTE: Pedro de Miguel Anasagasti

VOCAL: Antonio Cortes Rosello

VOCAL: Luis Pastor Pérez

VOCAL: Vicente Santonja Gisbert

SECRETARIO: José María Peña Sánchez

A mis padres y hermanas.
Por su apoyo incondicional.

"Uno no es lo que es por lo que escribe, sino por lo que ha leído"

Jorge Luis Borges

Agradecimientos

Ahora que finaliza esta etapa de mi vida, miro hacia atrás y me doy cuenta de toda la gente que ha hecho posible que esté escribiendo estas líneas de agradecimiento.

En primer lugar, me gustaría agradecer a mi familia su paciencia y apoyo todos estos años. Sin vosotros, nada hubiera sido posible. Quiero dedicaros especialmente el resultado de este largo trabajo.

Quisiera también agradecer a mis directores su ayuda durante toda la tesis. Gracias por haber estado siempre disponibles, a pesar de la distancia física.

Gracias a todos los compañeros del Departamento de Arquitectura por hacerme sentir como en casa. Me gustaría agradecer a Pedro sus consejos, ya que desde que comencé en el grupo de SSOO, me hizo ser consciente de la importancia de la tesis. A Chema, quisiera agradecerle su apoyo constante y haberse involucrado desde el principio en la consecución de la misma. Me gustaría agradecer a Fernando y Antonio ese empujón final; gracias por vuestras ideas y revisiones. Gracias Fran y Víctor, por vuestro compañerismo, por aguantarme y por hacerme sentir bien cada día. A José Luis, Victoria y Carmen su apoyo logístico en toda la burocracia que rodea el depósito de la tesis. A Santi, gracias por su ayuda con la escritura en LaTeX y por resolver mis dudas lingüísticas. A los compañeros del "Cine Jueves", gracias por convertir los jueves en viernes. A Ernes, gracias por tu ayuda. Y recuerda, siempre nos quedará Malvern. Gracias Ramón, Alberto y Marcus, por ayudarme en la parte técnica.

No quisiera olvidarme de todos aquellos amigos que han "sufrido" conmigo esta tesis. Esta tesis es un poco "vuestra". Gracias Carol, Luz, Carmen, "To", Chiqui, Guille, Luna, Indu, Salva, Marta, Yoli, *Rose*, Laura, Bea, Gustavo, Rosa, Rocío, Raquel, Vice, Alberto, Pili, por apoyarme en mis momentos bajos y por comprenderme.

Pido disculpas a todos aquellos que no menciono aquí, pero es imposible enumerar a todas las personas que me han aportado la energía necesaria para concluir la tesis.

A todos, mi eterno agradecimiento.

María de los Santos Pérez Hernández
Madrid, 8 de Abril de 2003

Resumen

La E/S constituye en la actualidad uno de los principales cuellos de botella de los sistemas distribuidos de propósito general, debido al desequilibrio existente entre el tiempo de cómputo y de E/S. Una de las soluciones propuestas para este problema ha sido el uso de la E/S paralela. En esta área, se han originado un gran número de bibliotecas de E/S paralela y sistemas de ficheros paralelos.

Este tipo de sistemas adolecen de algunos defectos y carencias. Muchos de ellos están concebidos para máquinas paralelas y no se integran adecuadamente en entornos distribuidos y clusters. El uso intensivo de clusters de estaciones de trabajo durante estos últimos años hace que este tipo de sistemas no sean adecuados en el escenario de computación actual.

Otros sistemas, que se adaptan a este tipo de entornos, no incluyen capacidades de reconfiguración dinámica, por lo que tienen una funcionalidad limitada.

Por último, la mayoría de los sistemas de E/S que utilizan diferentes optimizaciones de E/S, no ofrecen flexibilidad a las aplicaciones para hacer uso de las mismas, intentando ocultar al usuario este tipo de técnicas. No obstante, a fin de optimizar las operaciones de E/S, es importante que las aplicaciones sean capaces de describir sus patrones de acceso, interactuando con el sistema de E/S.

En otro ámbito, dentro del área de los sistemas distribuidos se encuentra el paradigma de agentes, que permite dotar a las aplicaciones de un conjunto de propiedades muy adecuadas para su adaptación a entornos complejos y dinámicos. Las características de este paradigma lo hacen a priori prometedor para abordar algunos de los problemas existentes en el campo de la E/S paralela.

Esta tesis propone una solución a la problemática actual de E/S a través de tres líneas principales: (i) el uso de la teoría de agentes en sistemas de E/S de alto rendimiento, (ii) la definición de un formalismo que permita la reconfiguración dinámica de nodos de almacenamiento en un cluster y (iii) el uso de técnicas de optimización de E/S configurables y orientadas a las aplicaciones.

Abstract

Nowadays, the I/O system is one of the major bottlenecks in general purpose distributed systems, because of the difference between the computing and I/O time. Parallel I/O is one of the proposed solutions to this problem. In this area, a great number of parallel I/O libraries and parallel file systems have been developed.

These systems have some drawbacks. Many of these ones are thought for parallel machines and are not suitable for distributed systems and clusters. The proliferation of these last environments make worse this disadvantage.

Other systems, which are based on clusters, do not include dynamic reconfiguration features and provide a limited functionality.

Finally, most of the I/O systems that use I/O optimizations, do not provide flexibility to the applications, trying to hide these features to such applications. Nevertheless, with the aim of increasing I/O operations performance, it is important to allow the applications to describe their access patterns and to interact with the I/O system.

On a different domain, the agents paradigm offers several characteristics to the applications, very appropriate for their adaptation to complex and dynamic environments. These features are promising for solving some of the problems on the parallel I/O field.

This thesis proposes a solution of the I/O problem through three different lines: (i) the usage of agents theory in high-performance I/O systems, (ii) the definition of a formalism that allows storage servers of a cluster to be reconfigured dynamically and (iii) the usage of configurable and application-oriented I/O optimization approaches.

Índice general

Capítulo 1

Introducción

Los sistemas de E/S han sido frecuentemente "marginados" en el mundo de la arquitectura, cuyos logros en el campo de los procesadores han ocultado y mantenido en la sombra toda la problemática del sistema de almacenamiento y recuperación de la información.

Un ejemplo de este abandono lo constituye el hecho de que *benchmarks* populares tales como SPEC hayan descalificado a *benchmarks* cuyas fases de E/S ocupan más de un 5 % del tiempo total de proceso.

No obstante, la importancia de los sistemas de E/S y de su rendimiento es hoy en día incuestionable. Para demostrar esto, simplemente basta con fijarse en los nombres utilizados para las distintas épocas de la informática [JCB02]. Al período transcurrido entre los años 60 y 80 se le denominó *época de la revolución de la computación*. Por el contrario, al período que comienza a partir de los años 90 se le denomina *era de la información*. Este cambio de mentalidad da un mayor protagonismo a la información y a los sistemas que se encargan de su almacenamiento y recuperación.

El aumento de la importancia de la información frente al procesamiento también ha sido recogido por el campo de la programación. La programación orientada a objetos [Boo91], que ha supuesto un cambio radical en dicha disciplina, da mayor prioridad a los datos que la programación clásica.

Por otro lado, las mejoras en los tiempos de acceso de los discos no han guardado proporción con el incremento del rendimiento de los procesadores, que ha mejorado más de un 50 % cada año. A pesar de que la densidad magnética media se ha incrementado de forma drástica [Hug02], superando incluso lo estimado por la ley de Moore y reduciendo los tiempos de transferencia del disco entre un 60 % y un 80 % cada año, el tiempo total de acceso sólo se ha reducido un 10 % cada año, debido a su dependencia de sistemas mecánicos. La ley de Amdahl afirma que el *speedup* logrado por los procesadores queda limitado por el componente del sistema más lento. De aquí, se concluye de nuevo que es necesario mejorar el rendimiento del sistema de E/S a fin de acercarlo al rendimiento de la CPU. Una de las formas de realizar esta tarea es llevar a cabo operaciones de E/S en paralelo, a través del uso de sistemas de ficheros paralelos.

Es de especial revelancia esta limitación para determinadas aplicaciones que gestionan grandes cantidades de datos y han de hacerlo de forma eficiente. En los últimos años han cobrado relevancia

pública los temas relacionados con las ciencias de la vida y el comportamiento humano y, por extensión, las aplicaciones asociadas a los mismos. El conocimiento del genoma humano, los progresos en la neurociencia o el desarrollo de la Inteligencia Artificial son algunos de los aspectos en los que se centra la actividad científica del siglo que acaba de comenzar, de tal modo que ya se empieza a hablar de *era del conocimiento*. Tal vez la distinción más clara entre la era del conocimiento y la era de la información sea el énfasis de la primera en el significado y semántica de dicha información. Pero es obvio que ambos conceptos, información y conocimiento, están intimamente relacionados, aunque no pocas veces se tiende a olvidar.

Es por tanto lícito pensar que los sistemas que se encargan de la recuperación y del almacenamiento de la información dependan de las aplicaciones que utilizan dicha información y que van a permitir explotar el conocimiento inherente en la misma. Existe una tendencia a diseñar e implementar los sistemas de almacenamiento sin tener muchas veces en cuenta las capas superiores por las que son utilizados, siendo los primeros totalmente transparentes a estas últimas. No obstante, parece razonable establecer determinados patrones de almacenamiento, que permitan optimizar el acceso a los datos dependiendo del dominio de la aplicación.

Por último, los avances realizados en el campo de las redes han provocado un cambio de escenario en los sistemas de E/S actuales, tendiéndose a utilizar nodos de E/S distribuidos, de forma análoga a las denominadas SAN (*Storage Area Network*).

Debido a las mejoras producidas en el hardware y sobre todo al decremento sufrido por los precios de los componentes informáticos, los clusters han aparecido como una alternativa muy atractiva en el campo de la computación paralela y distribuida.

En este marco y a fin de integrar distintas funcionalidades deseables en el sistema de almacenamiento se pueden utilizar diferentes tecnologías de sistemas distribuidos, entre las que se encuentra la tecnología de agentes, que destaca por su relativa juventud y por su adaptación a diferentes dominios. El uso de agentes también permite proporcionar otras características adicionales, tales como autonomia, reactividad y proactividad.

1.1. Estudio del problema

Los actuales avances en el diseño de procesadores junto con la proliferación de estaciones de trabajo de alto rendimiento y las mejoras en la tecnología de redes ha creado un enorme interés en el diseño y construcción de software para sistemas distribuidos y paralelos. Dentro de este interés, el diseño de sistemas operativos, y como parte de los mismos, el de sistemas de ficheros y almacenamiento para entornos distribuidos ha ocupado un lugar preferente.

Los sistemas de ficheros proporcionan una capa de abstracción a las aplicaciones que les permite explotar el sistema de E/S de una forma eficiente y sencilla. Por un lado, los sistemas de ficheros distribuidos permiten el acceso y la compartición de múltiples dispositivos de almacenamiento. No obstante, este tipo de sistemas quedan limitados por el acceso secuencial a cada servidor, lo que convierte a dichos servidores en un cuello de botella en el sistema. De este modo, el sistema de E/S limita el incremento de rendimiento logrado por las nuevas arquitecturas.

De cara a resolver toda la problemática que plantea la E/S, se han propuesto diferentes soluciones, entre las que cabe destacar el uso de sistemas de E/S paralelos. Actualmente, la E/S paralela se utiliza en diferentes tipos de aplicaciones, cada una de las cuales tiene requisitos muy diferentes. Por este motivo, han surgido diferentes bibliotecas de E/S paralela que ofrecen a los programadores de aplicaciones un conjunto de funciones de E/S altamente especializadas y que intentan obtener el máximo rendimiento y flexibilidad para cada clase de aplicación. Esta situación ha desembocado en una falta de estandarización de la E/S paralela. No existe una API estándar utilizada por todas las

bibliotecas de E/S. El resultado es una falta de portabilidad de las bibliotecas de E/S paralela. De hecho, MPI-IO es la única iniciativa real de intento de estandarización de la interfaz paralela de E/S. El principal problema de las bibliotecas de E/S paralela es que están concebidas fundamentalmente para el desarrollo de aplicaciones paralelas y no ofrecen una solución genérica para su uso en sistemas distribuidos. Por otro lado, los sistemas de ficheros paralelos operan independientemente de la aplicaciones ofreciendo una mayor flexibilidad y generalidad.

No obstante, los sistemas de ficheros paralelos existentes adolecen de algunos defectos y carencias, a saber:

- En general, se trata de soluciones "ad-hoc", que utilizan servidores propios, incompatibles con los de otros sistemas de ficheros. Además, el uso de servidores propios hace más complejo el sistema de E/S y complica su instalación y uso.

- Estos sistemas están especialmente pensados para máquinas paralelas y, en general, no se integran de forma adecuada en entornos distribuidos de propósito general. La proliferación de este tipo de entornos agrava aún más este inconveniente.

- La explotación de los sistemas de ficheros paralelos, así como de la computación paralela en general ha estado unida tradicionalmente a entornos formados por grandes computadores con una alta capacidad de procesamiento, pero a cambio de un alto coste.

- Los sistemas de ficheros paralelos no suelen estar concebidos para su uso en entornos de computación heterogéneos. El incremento de entornos de este tipo provoca que estos sistemas no suplan las necesidades actuales de computación.

- La mayoría de los sistemas de E/S que utilizan diferentes optimizaciones de E/S, tales como *caching*, *prefetching* o *hints*, no ofrecen flexibilidad a las aplicaciones para hacer uso de las mismas, intentando ocultar al usuario este tipo de técnicas. No obstante, para aplicaciones críticas en el tiempo o soluciones muy optimizadas sería interesante disponer de algún tipo de interfaz que permitiera adaptar dichas técnicas a su dominio concreto.

Además de estas desventajas, el escenario de computación actual ha cambiado en los últimos años, utilizándose de forma intensiva clusters de estaciones de trabajo, en lugar de los tradicionales supercomputadores especializados.

Como parte integrante de estos clusters se utilizan un conjunto de servidores de ficheros tradicionales y distribuidos, sin características de paralelismo, pero capaces de proporcionar la funcionalidad necesaria para acceder a los ficheros almacenados en los mismos. Estos servidores están ampliamente probados, por tratarse de servidores de uso extendido. Un ejemplo característico de este tipo de servidores lo constituyen los servidores NFS [SGK+85].

Dentro del área de los sistemas distribuidos se encuentra el paradigma de agentes, que permite abordar los problemas de una forma completamente nueva en el mundo de la computación. A pesar de que esta disciplina suele suscribirse dentro del área de la Inteligencia Artificial Distribuida (IAD), se ha utilizado en campos muy diversos. La principal ventaja de la tecnología de agentes es su aporte conceptual, que permite dotar a las aplicaciones de un conjunto de propiedades muy adecuadas para su adaptación a entornos complejos y dinámicos. Las características de este paradigma lo hacen a priori prometedor para abordar algunos de los problemas existentes en el campo de la E/S paralela.

1.2. Antecedentes y motivación de la tesis

La presente tesis surge a partir de la investigación iniciada por el Grupo de Sistemas Operativos del Departamento de Arquitectura y Sistemas Informáticos de la Facultad de Informática de la Universidad Politécnica de Madrid en el área de sistemas de E/S paralela. En este contexto, la primera línea de investigación realizada por dicho grupo consistió en el diseño y realización del sistema de ficheros ParFiSys [GCPdM99], que proporciona un conjunto de servicios de E/S para aplicaciones científicas y que permite explotar el paralelismo inherente en multicomputadores genéricos de paso de mensajes.

La experiencia en la realización del sistema ParFiSys se redirigió a otros aspectos dentro del mismo área de conocimiento, tales como el diseño e implementación de un sistema basado en técnicas de E/S colectivas para aplicaciones irregulares, denominado Compassion [NsPCC02] o el de un sistema de ficheros paralelo para aplicaciones multimedia denominado MiPFS [FGC00].

A partir de la experiencia de dicho grupo en la realización de sistemas de E/S, se planteó la conveniencia de desarrollar un sistema de E/S de propósito general para un entorno distribuido mediante el uso de servidores tradicionales y distribuidos existentes. Algunos investigadores del grupo pasaron a formar parte de la Universidad Carlos III de Madrid, consolidando el Grupo de Arquitectura y Computadores de dicha Universidad. Esta última línea de investigación se está desarrollando como colaboración entre ambos grupos de investigación pertenecientes a la Universidad Carlos III y la Universidad Politécnica de Madrid.

Este trabajo de investigación se ha materializado en dos líneas principalmente: (i) el desarrollo del sistema de ficheros Expand [GCPS02], que se centra en el diseño e implementación de la arquitectura de un sistema de ficheros paralelo que utiliza servidores NFS mediante el uso de técnicas tradicionales de E/S paralela y (ii) el diseño de una arquitectura multiagente para E/S en clusters, que se basa en el uso de la tecnología de agentes para proporcionar un alto rendimiento a las operaciones de E/S [PCG+03a].

La presente tesis se inscribe dentro de esta última línea, aunque ambas líneas comparten la arquitectura básica del sistema y tienen como objetivo común el desarrollo de una arquitectura que emplee servidores existentes.

1.3. Objetivos de la tesis

La falta de solución al problema de la E/S en sistemas distribuidos, y más específicamente en clusters, es la que motiva esta tesis, cuyo objetivo central es **demostrar la factibilidad de aplicar la teoría de agentes en el campo de la E/S paralela**. Para tratar de demostrar esta hipótesis de partida, se llevará a cabo el diseño e implementación de una arquitectura multiagente de E/S paralela para clusters, utilizando el paradigma de agentes, con la intención de proporcionar un alto rendimiento a la solución. El uso de agentes proporciona una serie de características que permiten, de una forma modular y extensible, añadir funcionalidad a un sistema software. Estas características son, entre otras, la autonomía, la reactividad y la proactividad.

La arquitectura planteada se basará en el uso de servidores de ficheros tradicionales o distribuidos, ampliamente utilizados, como repositorios de los datos, siendo la parte cliente la que se encargue de proporcionar paralelismo a la solución integral. Según se afirma en [CF96], "... el hecho de que los sistemas distribuidos no sean adecuados para los sistemas paralelos es lo que ha llevado al diseño de varios sistemas de ficheros paralelos ...". Esta tesis pretende conciliar ambos tipos de sistemas, permitiendo que los sistemas distribuidos puedan ser utilizados precisamente para la construcción de un sistema de ficheros paralelo, proporcionando características paralelas de E/S al acceso a servidores tradicionales o distribuidos.

Como primer prototipo del sistema, se emplearán servidores NFS. No obstante, el sistema deberá poder utilizar cualquier otro tipo de servidor tradicional o distribuido, si se implementa la interfaz adecuada con el sistema de ficheros objetivo de esta tesis doctoral.

Con la intención de resolver los problemas descritos de los sistemas de E/S actuales, utilizando la teoría de agentes, no empleada previamente en este tipo de sistemas, pero adecuada en el campo de los sistemas distribuidos, se pretende definir una nueva arquitectura de E/S de alto rendimiento en clusters. La propuesta planteada tiene como principales objetivos:

- Análisis de la **viabilidad del uso de la teoría de agentes en sistemas de E/S de alto rendimiento**.
- **Estudio formal de la reconfiguración dinámica** de nodos de almacenamiento en un cluster.
- Estudio de **técnicas que permitan proporcionar mayor interactividad a las aplicaciones respecto al sistema de E/S subyacente**, a fin de incrementar el rendimiento de dichas aplicaciones.

Para satisfacer los anteriores objetivos, se proponen los siguientes hitos:

- El diseño e implementación de una arquitectura de E/S paralela para clusters basada en el empleo de servidores heterogéneos existentes y ampliamente probados, tales como servidores NFS. Esta propuesta es diferente a la de la mayoría de los sistemas de ficheros paralelos, que se construyen desde cero, sin basarse en otras implementaciones y desarrollando tanto el cliente como el servidor. Esta arquitectura debe poder adaptarse a sistemas de almacenamiento actuales tales como las SAN.
- La definición de un formalismo denominado grupo de almacenamiento, que permita gestionar y configurar de una forma dinámica los servidores que forman parte de la arquitectura.
- La definición de técnicas de optimización de E/S configurables por parte de las aplicaciones.
- La utilización de la teoría de agentes como marco conceptual para el diseño y desarrollo de la arquitectura de E/S.

Como se ha mencionado anteriormente, uno de los objetivos de la arquitectura planteada es mantener la independencia de los servidores, de forma que no sea necesario implantar ningún módulo adicional en los mismos ni modificarse. Por tanto, el objetivo es construir un módulo cliente que permita acceder en paralelo a los datos distribuidos entre varios servidores convencionales. A este módulo se le denominará MAPFS (*MultiAgent Parallel File System*) [PGC01]. Los ficheros MAPFS deben distribuirse a lo largo de un conjunto de servidores, de forma que coexistan con particiones distribuidas y tradicionales. Esta característica hará que el sistema se integre adecuadamente en entornos distribuidos.

El acceso en paralelo a datos distribuidos tiene fundamentalmente dos ventajas:

1. Reduce el cuello de botella que constituye el acceso a servidores convencionales o distribuidos.
2. Mejora el uso de los recursos del sistema, debido a que la distribución de los datos entre diferentes servidores implica un mejor equilibrio de carga.

Por otro lado, la naturaleza heterogénea con que se pretende dotar a MAPFS permitirá el acceso a servidores con diferentes arquitecturas y sistemas operativos.

Además de estas características básicas, MAPFS debe permitir la adición de funcionalidades opcionales, tales como el uso de cache o tolerancia a fallos. Respecto a la primera de las propiedades, el

uso de almacenamiento intermedio o cache en el sistema de E/S es una de las soluciones que se han dado al problema de la E/S en sistemas distribuidos y paralelos, a fin de incrementar el rendimiento de dicha fase. En cuanto a la tolerancia a fallos, no ha sido añadida de forma tradicional a los sistemas de ficheros paralelos, aunque actualmente la mayoría de estos sistemas la tienen en cuenta ya incluso en las primeras fases de diseño. Estos son algunos de los servicios que se desean proporcionar al sistema de ficheros. Se desea que este conjunto de servicios se pueda extender con nuevas funcionalidades de una forma sencilla y flexible.

Para llevar a cabo el diseño e implementación de estos servicios, se hará uso del paradigma de agentes. Los agentes son primordialmente una herramienta de diseño y una abstracción que permite construir sistemas complejos. Son apropiados para la realización de la arquitectura propuesta, debido a su capacidad para enfrentarse a entornos dinámicos. Es por ello que se propone la división del sistema MAPFS en dos subsistemas, uno de los cuales constituya un sistema multiagente. El otro subsistema constituirá el núcleo del sistema de ficheros y contendrá la funcionalidad básica del mismo.

Se propone implementar los servicios de tolerancia a fallos y *caching* mediante el sistema multiagente asociado. Este subsistema también debe ser el responsable de planificar la recuperación de la información (*information retrieval*) [Voo94], dado que la misma debe estar distribuida entre diferentes nodos. Por tanto, las propiedades que debe proporcionar el sistema multiagente al sistema de ficheros son:

- Servicio de *caching* y/o *prefetching*.
- Tolerancia a fallos, en la forma de disponibilidad de los datos.
- Servicio distribuido de recuperación de información.

Otros servicios adicionales deben poder añadirse a través del mismo subsistema, que se encargará de dar soporte al sistema de ficheros. De este modo, todos los servicios quedan desacoplados del núcleo central.

Por otro lado, la mayoría de los sistemas de ficheros proporcionan una interfaz y acceso comunes a las aplicaciones, a pesar de sus diferentes requisitos de E/S. De cara a adaptarse a estas necesidades, MAPFS debe ofrecer una interfaz de configuración, que permita describir patrones de acceso e información adicional (*hints*) sobre los datos.

De este modo, MAPFS debe ofrecer una solución integral, al permitir que las aplicaciones, que forman parte del nivel superior, configuren el sistema de E/S con la finalidad de mejorar el rendimiento de la ejecución de las mismas. Para completar esta solución integral, es necesario permitir a las aplicaciones poder interactuar con la parte servidora de la arquitectura, pero sin modificar dichos servidores. De este modo, se utilizará el concepto de **grupo de almacenamiento**, que permita abstraer de forma lógica el concepto de grupo de servidores y llevar a cabo una gestión dinámica de los mismos. Para su correcto funcionamiento y mayor flexibilidad, el entorno distribuido en el cual se implante el sistema de ficheros, debe poder variar su topología de forma dinámica. El formalismo de grupos debe permitir modificar dicha topología, sin afectar al rendimiento de las operaciones de E/S.

1.4. Organización de la tesis

El trabajo se divide en tres bloques temáticos, cada uno de las cuales se divide a su vez en capítulos:

- El primer bloque recoge el estado actual de las áreas de investigación relacionadas con la presente tesis, a fin de encuadrar en el contexto actual dicho trabajo.

- El capítulo 2 analiza las diferentes tecnologías existentes en la computación distribuida y paralela, así como las nuevas tendencias dentro de este campo.
- El capítulo 3 describe la tecnología de agentes, como un nuevo paradigma de desarrollo de sistemas distribuidos. Esta tecnología aporta una serie de conceptos deseables para el desarrollo de subsistemas autónomos dentro de la arquitectura propuesta.
- El capítulo 4 presenta los sistemas de E/S actuales, haciendo énfasis en los sistemas de E/S distribuidos y paralelos.

▪ El segundo bloque constituye el núcleo de esta tesis, al presentar la solución ofrecida a la problemática descrita en este capítulo.

- El capítulo 5 muestra el diseño de la arquitectura MAPFS, como solución integrada que permite hacer frente a los problemas descritos en la sección 1.3.
- El capítulo 6 describe la estructura de un agente en el sistema y detalla el agente *proxy* como ejemplo de uso de este paradigma en la arquitectura MAPFS.
- El capítulo 7 analiza el formalismo de grupos de almacenamiento como abstracción lógica de los nodos de almacenamiento.
- El capítulo 8 describe diferentes optimizaciones realizadas por la arquitectura MAPFS de cara a incrementar el rendimiento de la fase de E/S de las aplicaciones. Además, define los perfiles de E/S como especificaciones de E/S de las aplicaciones, las cuales permiten detallar por parte de éstas la topología del sistema, los sistemas de agentes adicionales y las optimizaciones utilizadas.
- El capítulo 9 describe aspectos relacionados con la implementación del prototipo de la arquitectura propuesta.
- El capítulo 10 enumera y describe diferentes campos de aplicación de la arquitectura propuesta.
- El capítulo 11 muestra y evalúa diferentes escenarios que utilizan el sistema de E/S propuesto.

▪ El tercer y último bloque enumera las conclusiones y futuras líneas de trabajo.

- El capítulo 12 analiza las aportaciones realizadas por este trabajo y describe las posibles líneas de investigación y desarrollo que se derivan de esta tesis.

Parte I

ESTADO DE LA CUESTIÓN

Capítulo 2

Sistemas distribuidos y paralelos

2.1. Introducción

En sus inicios, los sistemas informáticos se componían de un computador central con una gran potencia de cálculo y muchos terminales *"tontos"* que se conectaban a él. Todas las aplicaciones se ejecutaban en el sistema central, utilizándose los terminales exclusivamente para la introducción de mandatos o instrucciones y la presentación de resultados. Esta configuración era poco flexible e insegura. Un fallo del computador central paralizaba totalmente el trabajo de todos los usuarios. Al ser dichos sistemas muy caros y difíciles de mantener, era impensable utilizar mecanismos de redundancia que evitaran que el *host* central fuera un punto único de fallo.

La aparición de equipos con una potencia de cálculo media pero a un precio asequible, las estaciones de trabajo o PC, hizo evolucionar el anterior escenario a otro en el que los sistemas informáticos se componen de numerosos equipos de baja y media potencia conectados entre sí en una red. Las redes permiten compartir los recursos entre todos los computadores que las componen, abriendo un nuevo abanico de posibilidades a las aplicaciones, pero introduciendo también nuevos problemas.

Esta nueva situación implica un cambio en el hardware, pero de forma paralela también una evolución del software. En esta línea, los sistemas operativos permiten controlar tanto recursos propios como de otras máquinas. Por su parte, las aplicaciones se ejecutan en diferentes estaciones de trabajo y son capaces de comunicarse entre sí, apareciendo las primeras aplicaciones distribuidas. Este término se extiende a los denominados sistemas distribuidos, que engloban tanto la parte software como la parte hardware.

Por otro lado, aparece el concepto de *paralelismo* como técnica que permite eliminar el cuello de botella que implica el hecho de utilizar un único elemento de computación. En el caso de que el elemento de computación sea un procesador, se habla de **procesamiento paralelo**. En caso de que el elemento de computación sea el sistema de E/S, se habla de **E/S paralela** y, de forma genérica, se denomina **computación paralela**. Este concepto va de la mano del concepto de sistemas distribuidos, ya que éstos últimos proporcionan el entorno adecuado bajo el cual se puede aplicar la técnica de paralelismo, aunque no el único. La sección siguiente analiza los diferentes sistemas de computación de alto rendimiento, que permiten proporcionar grandes prestaciones de procesamiento en la actualidad.

2.2. Sistemas de computación de alto rendimiento

La necesidad de entornos de alto rendimiento ha propiciado la aparición y establecimiento de diferentes sistemas de computación, capaces de proporcionar una mayor capacidad de cómputo y mejores prestaciones, entre los que se encuentran los sistemas distribuidos, a saber:

- MPP (*Massively Parallel Processors*): Se trata de un sistema que no comparte elementos de computación. Está formado por varios nodos (de un orden entre 100 y 1000 elementos), interconectados a través de una red de conexión rápida. Cada nodo ejecuta una copia del mismo sistema operativo.
- SMP (*Symmetric Multiprocessors*): Estos sistemas permiten que todos los procesadores que lo forman compartan básicamente todos los recursos del sistema: memoria, sistema de E/S, etc. Sólo se ejecuta una copia del sistema operativo.
- CC-NUMA (*Cache-Coherent Nonuniform Memory Access*): Los procesadores pertenecientes a un sistema de este tipo comparten la memoria del sistema completo, aunque el tiempo de acceso a cada una de las porciones de la misma no es homogéneo.
- Sistemas distribuidos: Cada nodo ejecuta una copia del sistema operativo, pudiendo ser éste diferente en cada uno de los nodos. Se distinguen por poseer un menor acoplamiento entre los diferentes nodos.
- Clusters: Un cluster se define como un conjunto de estaciones de trabajo interconectadas a través de alguna tecnología de red. Se caracterizan por estar formadas por estaciones de trabajo de alto rendimiento, conectadas a través de redes de alta velocidad. Esta definición no entra en conflicto con la de sistemas distribuidos, aunque se suele reservar el término cluster para aquellos sistemas que proporcionan a un bajo coste algunas de las características siguientes: (i) alto rendimiento; (ii) escalabilidad; y (iii) alta disponibilidad.

Todos estos sistemas permiten el incremento de rendimiento y el procesamiento paralelo. Las características que diferencian a todos estos sistemas aparecen en [HX98] (tabla 2.1).

La tendencia en computación paralela ha pasado en los últimos años del uso de grandes plataformas de supercomputación especializadas al uso de sistemas de propósito general, formados por componentes más desacoplados y también más baratos. El uso de clusters ha sido en este sentido una de las grandes revoluciones dentro del mundo del procesamiento paralelo.

2.3. Sistemas distribuidos

Desde un punto de vista general, los sistemas distribuidos permiten resolver algunos problemas de los sistemas centralizados. Para paliar las deficiencias de estos últimos, los sistemas distribuidos ofrecen las siguientes ventajas [Tan92]:

- Compartición de recursos: Las estaciones de trabajo permiten el acceso a sus recursos compartidos a través de la red.
- Reducción de costes: Las redes de estaciones de trabajo ofrecen una mejor relación precio/rendimiento que los grandes *hosts* o *mainframes*.
- Tolerancia a fallos: Los sistemas distribuidos permiten replicar recursos y procesos, a fin de proporcionar fiabilidad y disponibilidad al sistema. De este modo, el fallo de un nodo no implica el fallo del sistema global.

Característica	MPP	SMP	CC-NUMA	S. Distribuidos	Clusters
Número de de nodos	O(100)-O(1000)	O(10)-O(100)	O(10)-O(100)	O(10)-O(1000)	O(1000) o inferior
Complejidad del nodo	Grano fino o medio	Grano medio o grueso	Grano medio o grueso	Amplio rango	Grano medio
Comunicación entre nodos	Paso de mensajes / Variables compartidas en memoria distribuida (DSM)	Centralizado	DSM	Ficheros compartidos, RPC, paso de mensajes e IPC	Paso de mensajes
Espacio de direcciones	Múltiple	Único	Único	Múltiple	Múltiple o único
Copias y tipo de sistemas operativos	Varios sistemas operativos micro-kernel o monolíticos	Un único sistema monolítico	Muchos sistemas	Varias plataformas homogéneas	Varias plataformas homogéneas o micro-kernel

Cuadro 2.1: Taxonomía de sistemas de computación de alto rendimiento

- Mayor escalabilidad: Las aplicaciones distribuidas suelen ser más escalables, debido a que están concebidas para su implantación y ejecución en un entorno que consta de un conjunto variable de recursos.
- Adaptación a determinadas aplicaciones: Existen algunas aplicaciones que son inherentemente distribuidas.

En el otro sentido, los sistemas distribuidos originan nuevas una nueva problemática, que aún no ha sido resuelta:

- Tal vez la más significativa sea su alta complejidad, que demanda un nuevo tipo de software, también más complejo.
- Al entrar en juego el uso de una red de interconexión, se originan nuevos problemas, debido a que la red pasa a ser un elemento crítico. La latencia de la red y las posibles pérdidas de mensajes son algunos de estos problemas.
- La seguridad de la red es uno de los temas más controvertidos y más abiertos de los sistemas distribuidos, en el sentido de que aún quedan muchos aspectos por resolver en este terreno.

Las aplicaciones distribuidas siguen generalmente el modelo cliente/servidor. La motivación fundamental de este modelo, que justifica su existencia y utilización, es solucionar el problema de la coincidencia temporal de las diferentes partes implicadas en la comunicación. Este problema es complejo, debido a la diferente escala de tiempo por la que se rigen los distintos nodos que forman parte de la comunicación. El modelo cliente/servidor resuelve este problema desde el punto de vista conceptual, asignando un papel a cada una de las partes que intervienen, asociado a un comportamiento que debe seguir. Tiene la gran ventaja de ser aplicable en todos los ámbitos del problema, permitiendo así capturar la complejidad que presentan las interacciones entre aplicaciones distribuidas.

El cliente inicia la comunicación haciendo una petición al servidor y se queda esperando su respuesta. El servidor, por su parte, está siempre en ejecución, y su tarea básica es escuchar posibles peticiones. Cuando llega una petición la atiende, ejecutando las tareas necesarias y transmitiendo la

respuesta al cliente. Este modelo reduce la complejidad del problema de la comunicación entre aplicaciones a uno único, resolver la comunicación entre el cliente y el servidor, representando el cliente y el servidor cualesquiera entidades que se comuniquen. Una misma entidad puede ejercer el rol de servidor o cliente, dependiendo de la entidad o entidades con las que establezca comunicación.

Una de las características deseables de un sistema distribuido es la transparencia, que permite que el sistema se perciba como un sistema único y no como un conjunto de componentes independientes. Hay diversos tipos de transparencia:

- Transparencia de localización: Los usuarios o aplicaciones que utilizan el sistema no deben saber dónde se encuentran situados los recursos. Por este motivo, el nombrado de los recursos no debe hacer referencia a dicha localización.
- Transparencia de migración: Los recursos deben poder migrar libremente a otra localización sin que sus nombres tengan que cambiar.
- Transparencia de replicación: Los usuarios o aplicaciones que utilizan el sistema no conocen cuántas copias existen de un recurso. El sistema es libre de hacer copias de un determinado recurso. Evidentemente añadir esta característica supone un problema de coherencia entre las copias, que será necesario resolver.
- Transparencia de concurrencia: Múltiples usuarios o aplicaciones pueden compartir los recursos de forma automática como si los usaran en exclusividad.
- Transparencia de paralelismo: El sistema distribuido debe ser capaz de tomar ventaja del paralelismo del sistema global, sin que el usuario o la aplicación se percaten de ello.

La situación ideal es aquélla en la que el sistema es transparente respecto a todos los criterios enunciados.

De cara a implementar sistemas distribuidos, se utilizan los denominados entornos de desarrollo distribuidos. Éstos últimos consisten en un conjunto de herramientas que facilitan el desarrollo de sistemas distribuidos, proporcionando un conjunto de servicios y funcionalidades básicas y ocultando la complejidad de bajo nivel. Los entornos de desarrollo distribuidos se caracterizan por facilitar la implantación de soluciones distribuidas, cumpliendo los siguientes requisitos:

1. La comunicación sobre una red de conexión debe ser transparente para los usuarios; es decir, se debe ocultar la comunicación real entre los componentes del sistema distribuido.
2. Se deben poseer mecanismos de conversión adecuados para que distintos computadores puedan comunicarse.
3. Deben existir mecanismos de nombrado y de localización de servicios.

La tendencia actual en el campo de los sistemas distribuidos es el uso de *middlewares*. Un *middleware* consiste en una capa de software situada encima del sistema operativo y de sistemas de comunicaciones heterogéneos, que ofrece un conjunto estándar de servicios y una interfaz uniforme a las aplicaciones distribuidas [BMB+00], [ACC+00]. Estos sistemas se caracterizan por ser independientes tanto del fabricante, como del hardware y sistema operativo que les da soporte.

Se pueden considerar las llamadas a procedimiento remotos y los sistemas de ficheros en red como los primeros *middlewares*. Actualmente, el ámbito de entornos que se clasifican de esta forma es mucho más amplio, tomando un papel destacado los modelos de componentes distribuidos, tales como CORBA de OMG, DCOM de Microsoft o *Enterprise Java Beans* de Sun.

2.3.1. RPC (*Remote Procedure Call*)

A la hora de programar aplicaciones que se comuniquen a través de la red, se puede hacer uso de lo que Stevens denomina *programación de red explícita*. Los sockets o XTI [Ste98] constituyen tecnologías que pertenecen a esta taxonomía y son frecuentemente utilizados en la programación de aplicaciones distribuidas. Por contraposición, otra alternativa es la *programación de red implícita* [Ste99]. RPC [BN84] (Llamada a Procedimiento Remoto) forma parte de este tipo de programación. White [Whi75] es uno de los primeros autores que describe esta tecnología.

RPC utiliza la tradicional llamada a procedimiento, pero el proceso invocador, que se denomina cliente, y el proceso que contiene el procedimiento invocado, el servidor, pueden estar ejecutando en diferentes máquinas. Estas máquinas pueden utilizar diferentes formatos para los datos: los tipos de datos pueden ocupar distinto tamaño y la ordenación de los bytes puede ser también diferente (por ejemplo, ordenación *big-endian* frente a ordenación *little-endian*). Sun RPC utiliza XDR (*External Data Representation*) para la descripción y la codificación de los datos [Sri95b]. Por tanto, XDR es un lenguaje de descripción de datos y un conjunto de reglas para la codificación de los mismos.

Cualquier máquina en la que resida un servidor RPC debe ejecutar un programa denominado *port mapper* o *rpcbind*, que asocia los programas proporcionados por dicho servidor con distintos puertos TCP y UDP. Cuando un cliente RPC se inicia, contacta con el *port mapper* a fin de obtener el número de puerto del servicio deseado. Una vez obtenido, contacta con el servidor propiamente dicho, utilizando normalmente TCP o UDP.

En 1981 aparece Courier, que se considera el primer producto RPC desarrollado. La primera versión de Sun RPC apareció en 1985. Su nombre oficial es ONC/RPC (*Open Network Computing/Remote Procedure Call*), pero normalmente se denomina Sun RPC [Sri95a]. Las versiones iniciales de Sun RPC utilizaban sockets y los protocolos TCP o UDP. En los años 90, fue reescrito, utilizando TLI, el predecesor de XTI, permitiendo el uso de cualquier protocolo de red soportado por el kernel.

Probablemente la aplicación más extendida que utiliza Sun RPC es NFS (*Network File Systam*), el sistema de ficheros de Sun [NFS89], [NFS95]. Normalmente NFS no es construido a partir de las herramientas RPC estándar, sino que está optimizado. No obstante, la mayoría de los sistemas que soportan NFS, también soportan Sun RPC. Algunas implementaciones de NFS residen dentro del núcleo del sistema operativo por motivos de rendimiento.

Otro ejemplo de software que utiliza el sistema RPC y que es muy utilizado en los sistemas UNIX es NIS (*Network Information System*) [NIS02], que permite la compartición de datos de configuración y administración, mediante las denominadas bases de datos NIS.

A mediados de los años 80, Apollo diseñó su propio paquete RPC para competir con Sun, denominado NCA (*Network Computing Architecture*) [ZDL+89] y su implementación se llamó NCS (*Network Computing System*). Apollo fue adquirido por Hewlett Packard en 1989 y NCA paso a desarrollarse dentro de DCE (*Distributed Computing Environment*).

2.3.2. Componentes u objetos distribuidos

Debido a que el análisis y diseño orientado a objetos son herramientas para luchar contra la creciente complejidad del software, resulta lógico aplicarlo a las aplicaciones distribuidas, que se caracterizan precisamente por su complejidad.

Los componentes u objetos distribuidos constituyen una extensión de la filosofía de llamadas a procedimientos remotos en un entorno orientado a objetos. Los objetos pueden ser distribuidos a través de una red heterogénea y deben ser capaces de interoperar entre sí.

Los componentes tienen las siguientes propiedades principales:

- No deben estar sujetos a un determinado lenguaje. Componentes implementados en diferentes

lenguajes deben ser capaces de interaccionar entre sí.

- No deben estar sujetos a una determinado tipo de computador o sistema operativo, independizando los modos de representación de la arquitectura subyacente. Además, la localización física del componente debe ser transparente al sistema.
- Pueden utilizarse de una forma dinámica. Esto implica que existen mecanismos que permiten en tiempo de ejecución hallar componentes que proporcionen una determinada funcionalidad.

Por tanto, las tecnologías de componentes distribuidas ofrecen grados de libertad en el protocolo de comunicación, en el lenguaje de programación y en la arquitectura subyacente (hardware y sistema operativo).

De algún modo, los componentes suponen la implementación software del concepto de *plug-and-play*.

El concepto de componente ha supuesto un avance extraordinario en el desarrollo de los sistemas distribuidos, siendo ampliamente utilizados en la construcción de aplicaciones distribuidas.

La principal limitación de las tecnologías de componentes es que a pesar de que permiten resolver algunos de los problemas de bajo nivel de las soluciones distribuidas, no logran facilitar el diseño de la solución a un nivel más alto. Estas cuestiones quedan fuera de los objetivos de estas tecnologías, haciéndose necesario utilizar otros paradigmas, tales como el de **agentes** (véase capítulo 3).

Las tecnologías de componentes distribuidos más utilizadas son CORBA de OMG [OMG01], DCOM de Microsoft [Mic96] y *Enterprise Java Beans* de Sun [Sun01], [Sun02].

2.4. Computación paralela

La computación distribuida permite resolver algunos de los problemas de la computación tradicional. No obstante, la aparición de aplicaciones con unos requisitos de procesamiento y una demanda de velocidad cada vez mayores no encaja adecuadamente con la computación distribuida. De una forma aún más drástica, las aplicaciones denominadas GCAs (*Grand Challenge Applications*) [GCA] representan problemas no resolubles en una cantidad de tiempo razonable ni en los computadores actuales ni en los sistemas distribuidos. Aplicaciones que permiten modelar grandes estructuras de ADN o predecir el tiempo atmosférico constituyen ejemplos de este último tipo de aplicaciones. Finalmente, la experiencia demuestra que cualquiera que sea la velocidad de computación de los procesadores actuales, siempre habrá aplicaciones que requieran aún mayor potencia computacional.

Una forma de incrementar esta velocidad es paralelizar las aplicaciones con el objetivo de optimizar el acceso a los recursos.

La computación paralela no es una idea reciente. Ya Gill en 1958 escribía sobre programación paralela [Gil58]. Holland en 1959 describía "un computador capaz de ejecutar un número arbitrario de subprogramas simultanéamente" [Hol59]. En 1963, Conway describe el diseño de un computador paralelo y su programación [Con63]. Durante todo este tiempo, se ha realizado investigación en esta área. En 1996 Flynn y Rudd escribieron en [FR96]: "la continua tendencia hacia sistemas de alto rendimiento ... nos conduce a la siguiente conclusión: el futuro es paralelo".

El desarrollo de aplicaciones paralelas es una tarea compleja. En primer lugar, depende en gran medida de la disponibilidad de herramientas y entornos software adecuados. En segundo lugar, los desarrolladores de software paralelo deben enfrentarse a problemas que no aparecen en la programación secuencial, tales como la comunicación, la sincronización, el particionado y distribución de los datos, el reparto de carga, la tolerancia a fallos, la heterogeneidad de la arquitectura, los interbloqueos o las condiciones de carrera, entre otros. Algunos de estos problemas ya aparecen en los sistemas distribuidos.

Existen dos técnicas principales de programación paralela:

- La primera técnica se basa en el *paralelismo implícito*. En esta técnica, el usuario no especifica y, por tanto, no puede controlar la planificación del proceso y la distribución de los datos. Esta técnica es utilizada a nivel del lenguaje de programación y de los compiladores.

- La segunda técnica se basa en el *paralelismo explícito*. En esta técnica, el programador es responsable de paralelizar las aplicaciones, realizando la descomposición de las tareas, asignando los recursos a las diferentes tareas e implementando la estructura de comunicación.

El uso del paralelismo explícito suele proporcionar mejor rendimiento.
Los principales modelos y técnicas de programación paralela son:

- Compiladores paralelos: Esta técnica tiene una funcionalidad muy limitada, ya que se restringen a aplicaciones que exhiben comportamientos regulares, tales como aplicaciones que realizan cálculos en bucles. Se ha comprobado que esta técnica es bastante útil en el caso de aplicaciones que se ejecutan en multiprocesadores con memoria compartida o en procesadores vectoriales con memoria compartida, pero no en procesadores con memoria distribuida. El problema de este último tipo de sistemas es que tienen un tiempo de acceso a memoria no uniforme.

- Lenguajes paralelos: Los lenguajes paralelos han tenido poco éxito entre los programadores de las aplicaciones, debido a que los usuarios no suelen estar dispuestos a aprender un lenguaje nuevo exclusivamente para programación paralela. Como alternativa, se suelen utilizar lenguajes de alto nivel tradicionales y bibliotecas, que permitan extender las características de los primeros. Ejemplos de lenguajes paralelos lo constituyen SISAL [FCO90] y PCN [Fos91].

- Paso de mensajes: Esta técnica permite realizar programas paralelos eficientes para sistemas de memoria distribuida. El presente trabajo utiliza esta técnica, por lo que se analiza más detalladamente en la siguiente sección.

- VSM y DSM: VSM (*Virtual Shared Memory*) implementa un modelo de programación de memoria compartida en un entorno de memoria distribuida. Linda [ACG86] constituye un ejemplo de este tipo de programación. DSM (*Distributed Shared Memory*) [NL91] es la extensión del modelo de programación de memoria compartida en sistemas que físicamente no contienen memoria compartida. A diferencia de paso de mensajes, en un sistema DSM, un proceso que quiera obtener algún valor no necesita conocer su ubicación; el sistema lo localiza de forma automática. En la mayoría de los sistemas DSM, los datos compartidos pueden ser replicados a fin de aumentar el paralelismo y la eficiencia de las aplicaciones.

- Programación paralela orientada a objetos: La orientación a objetos proporciona un modelo de abstracción que facilita el diseño de las aplicaciones. Si en lugar de simples objetos se hace uso de objetos compartidos, es posible llevar a cabo una técnica similar al modelo de datos compartido pero orientada a objetos. La principal dificultad de esta técnica es que aún la comunidad científica no hace un uso extendido del paradigma de programación orientado a objetos, utilizando lenguajes tradicionales tales como C o Fortran. Algunos entornos orientados a objetos, tales como CC++ y Mentat, se han implementado para llevar a cabo una programación paralela [WL96].

2.5. Paso de mensajes

El paso de mensajes es un modelo de comunicación muy extendido en la computación paralela. Permite la implementación de programas paralelos en sistemas de memoria distribuida. La programación de este tipo de sistemas se puede realizar de tres formas diferentes:

1. Utilizando un lenguaje especial de programación paralela.
2. Utilizando una extensión de un lenguaje secuencial de alto nivel existente.
3. Utilizando un lenguaje secuencial de alto nivel existente y una biblioteca de paso de mensajes.

Respecto a la primera opción, tal vez el único lenguaje de programación especial que permite paso de mensajes sea *Occam*, diseñado para su utilización en el procesador *Transputer* [Inm84], [Inm86].

Algunos ejemplos de la segunda opción son el lenguaje *CC++*, que es una extensión del lenguaje *C++* y *Fortran M*, que es una extensión del lenguaje *FORTRAN*. Ambas extensiones permiten programación paralela en general [Fos95].

La tercera opción es la más común, debido a que utiliza lenguajes de alto nivel sin modificar. Para proporcionar las características de paso de mensajes utiliza bibliotecas que proporcionan dicha funcionalidad. Las dos tecnologías que se van a destacar en este apartado tanto por su importancia como su extendido uso son MPI y PVM, que constituyen proyectos desarrollados para proporcionar herramientas software que permitan implementar el modelo de paso de mensajes en estaciones de trabajo.

2.5.1. MPI

MPI (*Message Passing Interface*) [MPIc] es una interfaz estándar de paso de mensajes, que ha sido desarrollada por un consorcio de laboratorios de investigación, universidades y organizaciones comerciales, regentados bajo el nombre de *MPI Forum* [MPIb].

MPI es un paradigma de paso de mensajes, donde las tareas se comunican a través del envío de mensajes. El objetivo principal de MPI es lograr la portabilidad en un entorno distribuido, de forma similar al de un lenguaje de programación que permita la ejecución transparente de aplicaciones sobre sistemas heterogéneos. Al tratarse de una interfaz, MPI define la sintaxis y semántica de las operaciones que pueden utilizarse para llevar a cabo la comunicación por paso de mensajes. Todas las implementaciones existentes de MPI proporcionan dicha interfaz, manteniendo el comportamiento básico de las operaciones.

La primera versión de MPI [MPI94] pasó a ser estándar en 1994. Esta versión proporciona primitivas de comunicación punto a punto bloqueantes y no bloqueantes. Da soporte a tipos de datos derivados y permite realizar operaciones colectivas. Otras operaciones permiten establecer diferentes topologías de comunicación o modificar el entorno de los procesos.

MPI-2 [MPI97] fue desarrollado en 1997 y añade algunas novedades a la primera versión de MPI. Tal vez las características más destacables sean la gestión dinámica de procesos y el acceso a operaciones de E/S paralela. De hecho, MPI-2 proporciona un conjunto bastante rico de operaciones de E/S, englobadas bajo el nombre de MPI-IO [TGL99b], [CFF+95]. Estas utilidades están totalmente integradas con el resto de funciones de MPI.

MPI ha sido aceptado como un estándar de paso de mensajes por muchos fabricantes, entre los que se incluye DEC, Hitachi, IBM o Sun Microsystems. Algunas implementaciones son soluciones propietarias; otras, por el contrario, son portables a diferentes arquitecturas. Las implementaciones también se pueden caracterizar por ser de libre distribución o no. Además de implementaciones comerciales, existen un conjunto de implementaciones realizadas por centros de investigación o universidades, que

son de libre distribución, entre las que destacan MPICH [MPIa] o LAM-MPI [LAMb]. En [LAMa] se presenta una lista de implementaciones MPI disponibles.

2.5.2. PVM

PVM (*Parallel Virtual Machine*) consiste en un conjunto de bibliotecas que permite la ejecución de aplicaciones concurrentes o paralelas, en un entorno distribuido y heterogéneo de computadores con sistema operativo tipo UNIX.

Al igual que MPI, se basa en el mecanismo de paso de mensajes.

PVM fue desarrollado para un proyecto de computación distribuida de la NASA llamado *Beowulf* [BEO], que consiste en la creación de un cluster de computadores personales comunes mediante el uso de varios canales de comunicación que permitían tener una red de alta velocidad para la comunicación de los distintos componentes del cluster. Las características más importantes de PVM son su simplicidad, debido a su fácil instalación y su interfaz de programación sencilla, y su flexibilidad, debido a que se adapta a diferentes arquitecturas y distintas redes y, además, que se trata de un software de dominio público.

A diferencia de PVM, MPI consiste en una especificación que debe guiar las diferentes implementaciones que existen. Esto constituye una primera diferencia entre PVM y MPI, ya que la filosofía bajo la cual se construyeron ambos sistemas es diferente. No obstante, PVM también se puede considerar una especificación de una biblioteca para computación paralela, si se toma como tal la versión Oak Ridge de PVM [GBD+94a], [DGMP95]. De aquí surge la principal confusión a la hora de comparar MPI con PVM, ya que se tiende a comparar la especificación de MPI con la implementación de PVM [GL97]. Normalmente las especificaciones estándares especifican el mínimo número de características, ya que éstas deben ser de obligado desarrollo en las diferentes implementaciones. Por el contrario, las implementaciones ofrecen mucha más funcionalidad.

A pesar de que MPI es el estándar de la computación paralela, no se presenta como competencia de otras bibliotecas de este estilo, tipo PVM. Se pueden tener ambas tecnologías ejecutando sobre una misma plataforma hardware. De hecho, existe una implementación denominada *Unify* [CVRS94], que consiste en un subconjunto portable de operaciones pertenecientes a la especificación MPI, construidas encima de PVM. *Unify* permite que los programadores puedan desarrollar aplicaciones en las que convivan código MPI y código PVM. Se ha diseñado para que un programador pueda migrar de forma gradual las aplicaciones PVM a MPI.

PVM ofrece una máquina virtual, que se implementa mediante los denominados demonios PVM. Esta máquina virtual le proporciona un sistema operativo distribuido sencillo y útil. Además hace uso de interfaces especiales, para interactuar con otros sistemas de gestión de recursos.

A diferencia de MPI, PVM no ofrece un conjunto de operaciones de E/S paralela integrado dentro del resto del sistema [GL97]. No obstante, existen algunos proyectos como PIOUS [MS94], basados en PVM y que ofrecen todo un abanico de operaciones de E/S paralelas.

2.6. Clusters y sistemas de alto rendimiento

En los últimos años y dentro del campo de la computación paralela, se ha originado una tendencia a utilizar clusters de estaciones de trabajo, en lugar de los tradicionales supercomputadores especializados. Los principales motivos que han originado este hecho han sido [ACP95], [BFY96] que:

- El rendimiento de las estaciones de trabajo se ha incrementado de forma vertiginosa y es muy probable que se mantenga esta tendencia.

- Debido al uso de nuevos protocolos y tecnologías de red, el ancho de banda está incrementando y la latencia se está reduciendo, aunque no al mismo ritmo. Mientras que el futuro puede deparar redes con anchos de banda del orden de terabits por segundo, la latencia está dominada por la propagación de los retardos, hecho que no se espera que cambie significativamente.
- El precio de un cluster de estaciones de trabajo es más bajo.
- Debido a las características de las estaciones de trabajo, es más fácil integrar éstas en redes ya existentes que computadores paralelos especializados. Aún más, cada uno de los nodos puede mejorarse de forma sencilla, incrementando simplemente memoria o procesadores adicionales.
- Existe un mayor número de herramientas para estaciones de trabajo y además están más probadas.

De estas características, se deduce que las dos ventajas fundamentales de los clusters son su **escalabilidad** y su **bajo coste**.

De forma somera, un cluster se define como una colección de estaciones de trabajo o PC que están interconectados a través de alguna tecnología de redes y que muestran una imagen única del sistema[1] [BB99], [SB99].

Los clusters pueden clasificarse en diferentes categorías, dependiendo de diversos aspectos [BB99]:

- **Aplicación del cluster:** Según el dominio de aplicación, los cluster se clasifican en: (i) clusters de alto rendimiento (HP Clusters: *High Performance*), es decir, aquellos clusters que permiten lograr una gran eficiencia en el proceso de la información y (ii) clusters de alta disponibilidad (HA Clusters: *High Availability*), es decir, aquéllos que suelen utilizarse para aplicaciones críticas, en donde es necesario garantizar la disponibilidad de los recursos.
- **Hardware de cada nodo:** Dependiendo del tipo de nodo que se utilice, se pueden distinguir los clusters de PC (CoPs), también denominadas Pilas de PC (PoPs), clusters de estaciones de trabajo o *workstations* (COWs) y clusters de SMPs (CLUMPs).
- **Sistema operativo de cada nodo:** Según el sistema operativo que se ejecute en cada nodo, los clusters se diferencian en:
  - Clusters Linux, si ejecutan este sistema operativo (ejemplo: Beowulf).
  - Clusters Solaris (ejemplo: Berkeley NOW).
  - Clusters NT (ejemplo: HPVM).
  - Clusters AIX (ejemplo: IBM SP2).
  - Clusters Digital VMS.
  - Clusters HP-UX.
  - Clusters Microsoft Wolfpack.
- **Configuración de cada nodo:** La arquitectura y el sistema operativo de cada uno de los nodos condiciona el tipo de clusters. Según esta caracteristica, se pueden distinguir dos tipos de clusters: (i) clusters homogéneos, que son aquellos clusters en los cuales los nodos tienen una arquitectura similar y ejecutan el mismo sistema operativo y (ii) clusters heterogéneos, en los cuales los nodos tienen una arquitectura diferente y/o ejecutan diferentes sistemas operativos.

[1] SSI (*Single System Image*)

- **Niveles de clustering:** Según el número y ubicación de los nodos, se distinguen los siguientes tipos de clusters, de menor a mayor complejidad:
  - Clusters en grupo, donde el número de nodos oscila entre 2 y 99 y que se encuentran unidos por alguna tecnología de comunicación tipo SAN (*System Area Networks*), quedando confinados dentro de un mismo centro.
  - Clusters departamentales, cuyo número de nodos se encuentra entre 10 y 100 y los cuales pertenecen a un mismo departamento.
  - Clusters organizativos, que suelen estar formados por varios centenas de nodos y que pertenecen a una misma organización.
  - Metacomputadores nacionales, que permiten unir diferentes sistemas departamentales y organizativos y que tienen conexiones tipo WAN o basadas en Internet.
  - Metacomputadores internacionales, que incluyen desde miles hasta millones de nodos y que se basan en Internet para llevar a cabo la comunicación.

Las claves del éxito de los sistemas paralelos basados en un cluster son principalmente dos: las mejoras realizadas en estaciones de trabajo y en la red. Ambas han permitido incrementar el rendimiento de dichos componentes y la disponibilidad de APIs estándar de programación, que permiten beneficiarse de este incremento. Precisamente MPI es uno de los estándares más utilizados en una arquitectura de este tipo.

2.7. Computación grid

En la actualidad existe una gran cantidad de problemas cuya resolución excede la capacidad de los computadores. Todo esto ocurre a pesar de que un PC de hoy es más rápido que un supercomputador Cray de hace 10 años. A pesar de esto, en muchas ocasiones los computadores son infrautilizados. Esta circunstancia, y otras, hacen pensar que es posible construir un escenario en el cual todo este tipo de computación sea posible en un futuro, gracias a [Fos99]:

1. Mejoras en la tecnología: Se espera que los cambios en la tecnología VLSI y en la arquitectura de los procesadores incrementen la capacidad computacional en un factor de 10 en los próximos 5 años y un factor de 100 en los próximos 10.

2. Aumento del número de aplicaciones con accesos esporádicos a los recursos de computación: Debido a este motivo, se perfila un escenario en el cual se acceda a los recursos bajo demanda. Si se establecen mecanismos que permitan el acceso fiable y transparente a los recursos, desde el punto de vista de la aplicación es como si los recursos se dedicaran de forma exclusiva a la misma.

3. Incremento de la utilización de "tiempos muertos" de computación: La mayoría de los PC y estaciones de trabajo se encuentran frecuentemente "ociosos". De hecho, en los entornos académicos y comerciales la utilización de los mismos es aproximadamente del 30 % [ML91].

4. Mayor compartición de recursos computacionales: surgen problemas que pueden resolverse en colaboración y a través del acceso a múltiples recursos.

5. Nuevas técnicas y herramientas para resolución de problemas: Una gran variedad de técnicas puede mejorar la eficiencia o la facilidad con la que la computación se aplica a la resolución de problemas.

El término "grid" fue acuñado a mediados de los años 90 para denotar una determinada infraestructura de computación distribuida en el campo de la ingeniería y de la ciencia avanzada [FK99]. Desde entonces, se han realizado considerables avances en la construcción de dicha arquitectura [SWDC97], [BCF+98], [JGN99], que intenta dar salida al escenario descrito.

Al igual que en el campo de los agentes, no existe una perspectiva única de lo que es la computación grid. Una posible definición de lo que se denomina el "problema grid" [Fos01] consiste en "la compartición controlada y coordinada de recursos y el uso dinámico de los mismos en organizaciones virtuales". La compartición no se refiere exclusivamente al intercambio de ficheros, sino al acceso directo a computadores, datos, aplicaciones y otros recursos que pueden ser utilizados por un conjunto de aplicaciones que pueden colaborar a fin de resolver problemas. La compartición debe ser controlada a partir de reglas. Al conjunto de individuos o instituciones que definen dichas reglas se les denomina organizaciones virtuales (VO: *Virtual Organizations*).

No obstante, en torno a esta definición no consensuada surgen otras cuestiones en las que tampoco existe una visión única y uniforme, entre las que se encuentran la necesidad de "tecnologías grid" o la definición de aplicaciones grid.

2.7.1. Tecnología grid

La arquitectura grid necesita definir relaciones de compartición flexibles. Además requiere un sofisticado control sobre cómo se utilizan los recursos compartidos, sobre la delegación y la aplicación de políticas locales y globales. Las tecnologías distribuidas actuales no permiten cumplir los requisitos de dicha arquitectura. Las tecnologías de Internet permiten la comunicación y el intercambio de información entre diferentes equipos, pero no proporciona técnicas que permitan realizar un uso coordinado de los recursos situados en diferentes ubicaciones. Tecnologías de computación distribuida tales como CORBA permiten la compartición de recursos, pero sólo a nivel de una única organización.

Por tanto, es necesario el uso de una nueva tecnología, en la cual el problema central es la interoperabilidad, que asegure que se puedan iniciar relaciones de compartición entre diferentes partes, permitiendo la participación dinámica de nuevas entidades a través de diferentes plataformas, lenguajes y entornos de programación.

En primer lugar, una arquitectura grid necesita identificar y definir los protocolos que gobiernan la interacción entre los diferentes componentes. Estos protocolos van a permitir definir los mecanismos básicos por los cuales los usuarios de las VO y los recursos negocian, establecen y gestionan las relaciones de compartición.

En segundo lugar, los protocolos permiten definir servicios estándar que proporcionen determinadas capacidades a los participantes del VO. Los servicios se caracterizan por los protocolos que utilizan y por el comportamiento que implementa, de una forma similar a los servicios CORBA. *Globus Toolkit* [GLO] se ha convertido en el estándar "de facto" de los principales protocolos y servicios grid.

Por último y a fin de proporcionar abstracciones de programación del grid, es conveniente definir APIs (*Application Programming Interfaces*) y SDKs (*Software Development Kits*) que permitan facilitar la programación de aplicaciones grid y proporcionen portabilidad.

En los últimos años han surgido diferentes plataformas, tales como Globus [FK97], Legion [GWF+97] y UNICORE [AS99], que constituyen infraestructuras para la computación grid. Como ya se ha mencionado, Globus se está convirtiendo en el estándar de computación grid.

2.7.2. Aplicaciones grid

Basándose en algunos entornos de pruebas y en sistemas experimentales se han identificado cinco clases principales de aplicaciones grid:

- **Supercomputación distribuida:** Este tipo de aplicaciones utiliza grids con el objetivo de agregar múltiples recursos computacionales que permitan resolver problemas no resolubles en un único sistema.
- **Computación de alto rendimiento:** En este caso, el grid se utiliza para planificar un gran número de tareas independientes o ligeramente acopladas, de cara a aprovechar ciclos de procesador no utilizados, frecuentemente pertenecientes a estaciones de trabajo ociosas. El sistema Condor de la Universidad de Wisconsin [LLM88] permite gestionar un *pool* de cientos de estaciones de trabajo de diferentes universidades y laboratorios del mundo. Estos recursos se han utilizado en campos tan dispares como la simulación molecular de cristales líquidos o el diseño de motores diesel.
- **Computación bajo demanda:** Las aplicaciones bajo demanda usan el grid para solicitar recursos a corto plazo, que no pueden obtenerse de forma local.
- **Computación intensiva de datos:** El objetivo de este tipo de aplicaciones es obtener nueva información de datos que se encuentran geográficamente distribuidos entre diversas fuentes, tales como repositorios, bibliotecas digitales o bases de datos. En la actualidad, existe un gran número de aplicaciones que operan sobre grandes cantidades de datos. Diferentes aplicaciones de datos intensivas (*data-intensive applications* o *data grids*) se han utilizado en varios dominios, tales como la física [Hol00], la modelización climatológica [LEG+97], la biología [YFH+96] o la visualización [FL99].
- **Computación colaborativa:** Las aplicaciones colaborativas tienen como objetivo lograr la interacción entre humanos, de forma que las tareas se realicen de acuerdo a esta colaboración. Muchas de las aplicaciones colaborativas permiten la compartición de recursos tales como ficheros o simulaciones.

Esta tesis está relacionada con la gestión de recursos en sistemas paralelos y distribuidos. Por tanto, tiene una estrecha relación con la problemática grid y más concretamente con las aplicaciones grid que tienen un uso intensivo de datos. Actualmente, existen diferentes sistemas que ofrecen servicios para el acceso a recursos de un *data grid*. El acceso a recursos heterogéneos con diferentes interfaces y funcionalidades se resuelve, en la mayoría de los casos, por medio de nuevos servicios que ofrecen un acceso uniforme a diferentes tipos de sistemas. Ejemplos de este tipo de sistemas lo constituye Globus [VTF01], *Storage Resource Broker (SRB)* [BMRW98], DataCutter [BFK+00], DPSS [TLJ+99] y BLUNT [MR00]. Todos estos sistemas utilizan la replicación para mejorar el rendimiento de las operaciones de E/S y la disponibilidad del sistema.

Capítulo 3

Agentes

3.1. Introducción

¿Qué es un **agente**? Responder a esta pregunta no es una tarea sencilla u obvia. No existe una definición precisa de un agente, al igual que no existe una definición comunmente aceptada para el término **inteligencia artificial**, área en la que también de forma parcial, muchos autores suscriben el concepto de agente. De hecho, existen muchas definiciones de los agentes, dependiendo de los autores y sus intereses. Aún sin tener una definición exacta del concepto, sí se tiene en el conjunto de investigadores una percepción común del agente como un elemento que se comunica con su entorno de forma significativa, casi definitoria del mismo. Por ello, se presenta como interesante determinar cómo se pueden comunicar entre ellos y con su entorno, así como los medios que deben usar. Dado el aspecto androide que se le da, parece lógico que la comunicación se lleve a cabo mediante un lenguaje. Pero ¿este lenguaje debe ser universal, común a todos los agentes?, ¿debe poder modelar situaciones reales en base a la abstracción de conceptos como se deduce de las capacidades androides que se imputan a los agentes?, ¿debe permitir el conocimiento de nuevos elementos en un entorno dinámico y variable?.

A estas preguntas intenta responder este capítulo, intentando establecer el estado de la cuestión tanto en los aspectos básicos como en el concepto de la comunicación y de la cooperación entre diferentes agentes.

3.2. ¿Qué es un agente? Conceptos básicos de los agentes

Para encontrar la definición de una entidad debemos determinar una serie de características que la identifiquen y diferencien de todas las demás. Una característica común de la literatura sobre agentes es la falta absoluta de una definición formal del concepto de agente y el hecho de que siempre se intenta definir, utilizando un adjetivo calificativo del mismo, tras el que se esconde esta grave falta o deficiencia. Por otra parte, este adjetivo suele conllevar una serie de atributos determinantes que distorsionan totalmente el objetivo primario de determinar qué es un agente. Por todo ello, suele ocurrir que la concepción de la esencia de un agente viene dada a posteriori, basada en la experiencia

obtenida del trabajo realizado para la construcción del software necesario para lograr el objetivo planteado. Esto no es muy adecuado, ya que lo ideal es comprender un concepto antes de utilizarlo y no a la inversa, aunque bien es cierto que en muchas ocasiones el proceso de comprensión de un concepto es un proceso iterativo y que se realimenta con la etapa de utilización del mismo.

Por estos motivos, la definición de un agente no es algo inmediato, sino que consiste en un ejercicio de análisis de diversas fuentes bibliográficas. De entre la oferta existente de aquellos expertos que trabajan sobre el tema se puede observar que la definición de un agente es acorde con la investigación que realizan, incluso con el interés directo de la misma. Y la mayoría, como ya señalabamos previamente, se basan en características del agente para definirlo.

A continuación se enumeran algunas de las definiciones de agentes más revelantes, tanto por su originalidad como por su alcance.

Franklin y Graesser definen un agente, basándose en el concepto de **autonomía** [FG96]: "Un agente autónomo es un sistema con capacidad de autonomía, con propuestas de acciones en un mundo real".

Wooldridge y Jennings enfocan la definición de un agente basándose en sus propiedades [WJ95a]: "un agente es un sistema hardware o software que consta de las siguientes propiedades:

- Autonomía: un agente opera sin intervención directa de humanos u otros, y tiene un determinado control sobre su acciones y su estado interno;
- capacidad de relación (**social**): un agente interactúa con otros agentes (y posiblemente con humanos) a través del uso de algún tipo de lenguaje de comunicación de agentes.
- **Reactividad**: un agente es capaz de percibir el entorno y responder en un tiempo adecuado a los cambios que ocurren en el mismo.
- **Proactividad**: un agente no actúa simplemente en respuesta a su entorno, sino que es capaz de tomar la iniciativa a fin de lograr los objetivos".

Rao y Georgeff [RG95] definen los agentes como entidades racionales que poseen actitudes mentales: creencias, deseos e intenciones, definiendo lo que denominan arquitectura BDI (*Belief, Desire and Intention*). Las creencias consisten en el conocimiento que el agente tiene sobre sí mismo y su entorno. Los deseos son los objetivos que un agente desea satisfacer a largo plazo. Las intenciones son los objetivos a corto plazo, es decir, los que en cada momento el agente intenta llevar a cabo. En base a dicha arquitectura, definen todo un modelo lógico, que constituye un modelo de comportamiento de agentes autónomos.

Russell y Norvig, que se dedican al uso práctico de la Inteligencia Artificial en entornos reales, definen un agente en tres pasos [RN94], ordenados por un orden creciente de complejidad:

- Agentes genéricos: "Un agente es algo que puede ser visto como un perceptor de su entorno a través de sensores y que actúa sobre dicho entorno a través de actuadores (que modifican este entorno)". Esta definición es realizada en un sentido muy amplio y muchos elementos software podrían considerarse agentes si nos ajustamos estrictamente a dicha definición.
- Agentes racionales: "Un agente racional ideal realiza cualquier tarea a fin de maximizar su rendimiento, y en base a evidencias que le proporciona la secuencia de percepciones y cualquier otra fuente de conocimiento que el agente tenga".
- Agentes autónomos: "Un agente es autónomo siempre que sus acciones y elecciones dependen de su propia experiencia, en lugar del conocimiento del entorno que le ha proporcionado el diseñador originalmente."

Pattie Maes, pionera en el estudio de agentes, [Mae90b] pone el epíteto de autónomos a los agentes y los define de la siguiente forma: "Agentes autónomos son los sistemas computacionales que se desarrollan sobre entornos complejos y dinámicos, conociendo, actuando de forma autónoma sobre éste y trabajando para lograr un conjunto de metas para las que fue diseñado".

Virdhagreswaran en su línea de trabajo con agentes **móviles** en el campo de la Inteligencia Artificial define un agente como "el término usado para representar dos conceptos ortogonales. El primero es la capacidad de los agentes para su ejecución autónoma. El segundo es la capacidad para realizar razonamientos dentro de su ámbito". Por tanto, de nuevo se basa en la característica de autonomía y además enfoca la definición en la capacidad de razonamiento o **inteligencia** de los agentes.

Barbara Hayes-Roth [HR95] se basa en la característica de la inteligencia para definir un agente, concretamente un agente inteligente: "Los agentes inteligentes están continuamente realizando tres funciones:

- Percepción dinámica de las condiciones de su entorno.
- Acciones que afectan a las condiciones de su entorno.
- Razonamiento para interpretar lo percibido, resolver problemas, prever interferencias y determinar actuaciones".

Otros autores enfatizan otras características opcionales de los agentes, como son la movilidad [WP99],[Kna96] o la cooperación. Esta última característica, por ser de especial relevancia en este trabajo, se analiza más detalladamente.

Todas estas definiciones en alguna medida contribuyen a tomar conciencia del término agente y a ser capaces de distinguir un agente de lo que en términos estrictamente técnicos podría denominarse simplemente proceso.

Tal vez la única certeza que tenemos sobre los agentes es que han constituido un hito en el mundo de la Inteligencia Artificial.

3.2.1. Agencias

Los agentes, debido a su carácter social, comparten con los humanos algunas características. Una de ellas es la necesidad de "habitar" en lugares específicos, con los que se comunican para determinar sus características y posibilidades. Estos lugares o entornos se denominan **agencias** y son imprescindibles para el desarrollo de los agentes.

Un agente necesita estar ubicado en un lugar o entorno que le es propicio, que es lo que se denomina agencia. A fin de conocer su entorno, el agente debe utilizar un conjunto de sensores adecuado.

3.2.2. Taxonomía de agentes

La clasificación de los agentes se puede llevar a cabo de acuerdo con las características de los agentes, según sus múltiples peculiaridades:

- Reactividad o respuesta a estímulos exteriores y actuación consecuente que cambia el entorno.
- Proactividad o capacidad para tomar la iniciativa y actuar en pro de la consecución de unos objetivos.
- Autonomía, es decir, condición de una entidad que de nadie depende en ciertos aspectos.
- Persistencia o continuidad en el tiempo.

- Comunicación y sociabilidad, es decir, capacidad de comunicarse con otros agentes.
- Capacidad de aprendizaje y adaptación al medio.
- Movilidad, es decir, capacidad de transporte entre diferentes máquinas.
- Personalidad y estados emocionales (características androides).

De esta forma, se han catalogado diferentes tipos de agentes según la existencia o no de las anteriores características, de los cuales los más conocidos o estudiados son:

- Agentes autónomos, [FG96], [MPT94], si poseeen la característica de autonomía.
- Agentes inteligentes, [KJ98], [CKL94], [WJ95a] si poseen la capacidad de aprendizaje y adaptación al medio.
- Agentes sociales, [Fon93], [Wer89] si son capaces de comunicarse con otros agentes.
- Agentes móviles, [Try99], [WP99], [Whi95], [Bau99], [Kna96] si poseen la característica de movilidad.

También se pueden clasificar por su función en agentes de regulación, planificación o adaptación [Bru91].

Se pueden realizar muchas otras clasificaciones de acuerdo a las tareas que van a ejecutar. De esta forma, los virus pueden llegar a clasificarse como agentes de software, en contraposición con los de Vida Artificial[1] [Bel92] y como sistemas de computación en contraposición con los biológicos.

3.3. Origen de los agentes

La programación orientada a objetos representó en su día un gran salto adelante con respecto a la programación basada en procedimientos. Constituyó una completa revolución en el campo de desarrollo de aplicaciones, al permitir beneficiarse de dos conceptos que se complementan y se realimentan: *abstracción* y *reutilización*. El desarrollo de los componentes distribuidos supuso la aplicación de esta metodología en un entorno distribuido. La teoría de agentes va un paso más allá en el procesamiento distribuido permitiendo la interacción dinámica de componentes autónomos y heterogéneos.

Por tanto, los agentes se pueden considerar el siguiente paso evolutivo en la programación de sistemas, precediéndoles los sistemas orientados a objetos. En este sentido, los sistemas cada vez se encuentran más próximos al lenguaje natural y, en el caso de los agentes, a las características humanas. De hecho, como hemos visto previamente, es bastante común en Inteligencia Artificial caracterizar un agente utilizando nociones mentales, tales como conocimiento, creencia, intención y obligación; e incluso algunos investigadores hablan de agentes *emocionales* [WJ95a].

Pero un agente no es un objeto ni viceversa. Ambas entidades comparten el concepto de abstracción. Pero a diferencia de los objetos, los agentes se caracterizan por ser *autónomos* y depender del *entorno* en el que se encuentran situados. Por otro lado, los objetos dependen de su *estado* (atributos) y de los elementos externos que invocan sus métodos.

Además, una de las características fundamentales que distingue los agentes de otras abstracciones software es el concepto de *continuidad temporal* [FG96], de forma que un agente se encuentra activo de una manera continua en espera de estímulos que le lleguen del exterior y en respuesta a los mismos, lleva a cabo acciones dependiendo de determinados factores de su entorno.

[1] Dentro del área de Vida Artificial, se estudian los mundos artificiales. Éstos incluyen aplicaciones lúdicas, que contienen mundos donde agentes nacen, crecen, se alimentan, se reproducen y mueren, y otras aplicaciones, que, utilizando básicamente las mismas técnicas, tratan de reproducir características de sistemas vivos

3.4. Los agentes en la práctica

Como otros conceptos en informática, el concepto de agente queda muy difuso y es dificil de delimitar su contorno. No obstante, en este campo trabajan numerosos equipos obteniendo resultados cada vez mejores. Por ello, no debería ser un problema ponerse de acuerdo en un punto tan importante así como básico como es la terminología a emplear. El estudio de los agentes avanza en tres frentes comunes y diferentes:

- **Teoría de agentes:** esencialmente especificaciones sobre su conceptualización, tanto de los agentes como de las propiedades que debe tener, y de la manera en que se deben formalizar y representar ambos.
- **Arquitecturas de agentes:** representa el paso de la especificación a la implementación. Este campo permite la busqueda de las estructuras hardware y software más apropiadas para dicha implementación.
- **Lenguajes de agentes:** el tercer campo de estudio es el de los lenguajes para la comunicación de los agentes. Debido a que los agentes tienen un carácter social y son capaces de comunicarse, es necesario emplear un lenguaje para la comunicación entre dichas entidades. Esta área se encarga del desarrollo de dicho lenguaje.

3.5. Teoría de agentes

Las teorías de agentes son un conjunto de especificaciones que permiten conceptualizar los agentes. Estas teorías, en definitiva, son formalismos que permiten representar las propiedades de los agentes. A través del uso de estos formalismos, se desarrollan teorías que determinan las propiedades deseables de los agentes. Además, una teoría de agentes debe ser capaz de representar los aspectos dinámicos relacionados con los agentes. Por lo que una teoría de agentes completa debe definir las relaciones existentes entre los diferentes atributos de los agentes.

Los agentes suelen considerarse como sistemas intencionales. Fue el filósofo Daniel Dennett quién acuñó este término [Den78], [Den87]. Estos sistemas se caracterizan por estar formados por creencias, deseos e intenciones. De cara a modelizar estos sistemas, se han desarrollado diferentes teorías que se recogen en [WJ95a]. A continuación se describen cada uno de estos modelos.

3.5.1. Teoría de conocimiento y acción

Moore [Moo90] fue uno de los pioneros en el uso de la lógica en el campo de los agentes. Moore estudió lo que denominó "pre-condiciones de conocimiento para acciones", es decir, aquellas cuestiones que un agente necesita conocer a fin de llevar a cabo una acción. Este formalismo permite a un agente alcanzar una determinada meta a partir de información incompleta. Algunas críticas, así como mejoras a la teoría se recogen en [Mor89] y [Les89].

3.5.2. Teoría de intención

Una de las teorías más influyentes en el área de los agentes es la denominada teoría de intención, que desarrollaron Cohen y Levesque [CL90a], [CL90b].

A partir del análisis que hizo Bratman [Bra87], [Bra90], Cohen y Levesque identificaron las propiedades que debe satisfacer una teoría de la intención. A partir de estos criterios, aplicaron una solución de dos capas para la formalización de la intención:

1. Construyeron una lógica para agentes racionales, definiendo operadores modales básicos y sus relaciones.

2. Sobre esta capa, introdujeron una serie de constructores derivados. La intención es uno de estos constructores.

Singh realizó una crítica a la teoría de intención en [Sin92].

3.5.3. Modelo BDI

El trabajo realizado por Cohen y Levesque habla sólo de dos comportamientos básicos de los agentes: las creencias y las metas. La intención se define en esta teoría en términos de estos comportamientos básicos. Por el contrario, Rao y Georgeff crearon un entorno basado en tres primitivas, las creencias (*beliefs*), los deseos (*desires*) y las intenciones (*intentions*), desarrollando el modelo BDI [RG91b], [RG91a], [RG92a], [RG93], [RG95], [Rao96]. El modelo BDI está basado en un modelo de ramificación a través del tiempo (*branching-time model*) [EH86], en el cual la exploración de las intenciones se lleva a cabo a través de ramas construidas a lo largo del tiempo. Rao y Georgeff también consideraron la posibilidad de añadir planes sociales a su formalismo [RG92b], [KLR+92].

Este modelo ha sido ampliamente utilizado. Ejemplos de arquitecturas que lo utilizan son IRMA (*Intelligent Resource-Bounded Machine*) [BIP88] y PRS (*Procedural Reasoning System*) [GL87].

3.5.4. Modelo de Singh

Singh desarrolló un modelo bastante complejo, que incluye una familia de modelos lógicos para la representación de intenciones, creencia, conocimiento y comunicación en un entorno tipo *branching-time* [Sin90], [Sin91a], [Sin91b], [SA91]. [Sin94] recoge un compendio y una extensión de todos estos artículos.

3.5.5. Modelo de Werner

Werner [Wer88b], [Wer89], [Wer90], [Wer91], construyó un modelo general de agentes, basándose en un amplio abanico de campos, tales como economía, teoría de juegos, teoría de autómatas y filosofía. Werner hizo énfasis en el aspecto social y cooperativo de los agentes [Wer88a], que colaboran formando sistemas multiagente (véase sección 3.9).

3.5.6. Modelización de sistemas multiagentes

Wooldridge [Woo92], [WF92], desarrolló en su tesis doctoral una familia de modelos lógicos que permiten representar las propiedades de los sistemas multiagentes. La principal diferencia de este modelo frente a las técnicas anteriores es que éstas últimas tienen como objetivo el desarrollo de un entorno general para teoría de agentes. Por el contrario, este modelo consiste en la construcción de formalismos que pueden ser utilizados en sistemas multiagente reales, donde se necesitan protocolos de cooperación.

3.6. Arquitecturas de agentes

Se puede definir una arquitectura como la porción de un sistema que proporciona y gestiona los recursos de un agente. La arquitectura constituye la parte práctica de la agencia, ya que describe

aquellos aspectos relacionados con la construcción de sistemas que permiten a un agente mostrar la conducta enunciada en las teorías de los agentes.

Maes [Mae91] define una arquitectura de agentes como una metodología particular para la construcción de agentes, que especifica cómo un agente puede descomponerse en un conjunto de módulos y cómo estos módulos deben interaccionar a fin de proporcionar respuesta a la forma en que los datos de entrada y el estado interno de los agentes determinan las acciones y los futuros estados del agente.

Por su parte, Kaelbling [Kae91] afirma que una arquitectura de agentes consiste en una colección específica de módulos software y/o hardware, diseñados típicamente como cajas con flechas que describen los datos y el control de flujos entre los módulos. También añade que una forma más abstracta de ver una arquitectura sería como una metodología general que permite el diseño modular de tareas particulares.

En todo caso, las arquitecturas de agentes pueden ser clasificadas basándose en distintos criterios. La bibliografía de agentes muestra dos diferentes clasificaciones de las arquitecturas:

1. Las arquitecturas de agentes pueden clasificarse como arquitecturas **horizontales** o **verticales** [MPT94]. En las primeras, todas las capas tienen acceso a sensores y actuadores. Por el contrario, en las arquitecturas verticales sólo la capa más baja tiene acceso a dichos elementos. Las arquitecturas horizontales tienen como ventaja el paralelismo entre capas y como desventaja la necesidad de tener un gran conocimiento de control para llevar a cabo la coordinación entre ellas. Por su parte, las arquitecturas verticales reducen este control a cambio de una mayor complejidad de la capa más baja.

2. Las arquitecturas de agentes también pueden clasificarse según el tipo de procesamiento empleado [WJ95a] en arquitecturas **deliberativas**, **reactivas** e **híbridas**. A continuación se analizan más detalladamente estos tipos de arquitecturas.

3.6.1. Arquitecturas deliberativas

Este tipo de arquitecturas tienen como base la Inteligencia Artificial Simbólica y concretamente los trabajos realizados por Newell y Simon [NS76], según los cuales un sistema de entidades físicas o símbolos capaz de manipular estructuras **simbólicas** puede mostrar una conducta inteligente. A partir de esta hipótesis, aparece el concepto de *agente deliberativo* [GN87]. Estos agentes toman los fundamentos de la teoría clásica de planificación en Inteligencia Artificial [JHD90][RK94], [SMD94], según la cual a partir de un estado inicial, un conjunto de planes y un estado final como objetivo, el agente deliberativo construye los pasos a seguir.

Se pueden distinguir dos tipos de arquitecturas deliberativas [WJ95a], las **arquitecturas intencionales** y las **arquitecturas sociales**.

Las arquitecturas intencionales son aquéllas que permiten hacer razonamientos en base a creencias e intenciones. Ejemplos destacados de arquitecturas intencionales son las arquitecturas basadas en los modelos BDI [HS96].

Las arquitecturas sociales [MCd96] incluyen agentes sociales que mantienen modelos de otros agentes y en base a los cuales son capaces de razonar. Se pueden dividir en dos grupos. El primer grupo consiste en agentes intencionales con capacidad de razonamiento sobre la existencia de otros agentes. Ejemplos de arquitecturas de este tipo lo constituyen GRATE [Jen92], [JML$^+$92], COSY [BS92], [Had93], [HS96] y DA-SoC [HYGO94]. El segundo grupo consta de aquellas arquitecturas clásicas que se han centrado en la cooperación, dejando en segundo plano las intenciones de los agentes. ARCHON [Wit92], [CN96], IMAGINE [Hau94], [Ste96] y Coopera [SAvL96] constituyen ejemplos de este segundo tipo.

3.6.2. Arquitecturas reactivas

Las arquitecturas reactivas [Fer96] constituyen una alternativa frente a las arquitecturas deliberativas, ya que no utilizan un enfoque simbólico, sino conductista, mostrando un comportamiento estímulo-respuesta. Estas arquitecturas no utilizan un modelo del mundo a partir del cual construir los planes, sino que utilizan un conjunto de sensores, que activan una serie de actuadores bajo determinadas condiciones.

Brooks fue una de las voces más críticas contra el paradigma simbólico, afirmando que no es necesario el uso de una representación explícita del modelo para que una arquitectura muestre un comportamiento inteligente [Bro86], [Bro90], [Bro91a], [Bro91b]. De hecho, propuso la llamada *arquitectura de subordinación (subsumption architecture)*. Esta arquitectura se encuentra compuesta por un conjunto de niveles que siguen una determinada conducta. Cada nivel está formado por una red de máquinas de estado finitas. En base a esta arquitectura, Brooks construyó un gran número de *robots*. Steels [Ste90] llevó a cabo un trabajo similar al construir un sistema que simulaba la exploración de Marte (*Mars Explorer*), haciendo uso de agentes basados en este tipo de arquitectura.

Chapman y Agre [CA86] también exploraron alternativas al paradigma de planificación de Inteligencia Artificial. Observaron que la mayoría de las actividades son rutinarias y no requieren razonamiento abstracto nuevo, ya que una vez aprendidas, pueden llevarse a cabo de la misma forma, con pocas variaciones. Basándose en esto, propusieron el sistema PENGI [AC87], que utiliza una estructura de bajo nivel, similar a un circuito digital, que lleva a cabo actualizaciones periódicas, a fin de permitir tratar nuevas clases de problemas.

Rosenschein y Kaelbling [Ros85], [RK86], [KR90], [Kae91] propusieron una arquitectura reactiva en la cual un agente se especifica de forma declarativa. Su especificación es compilada y convertida en una máquina digital, que satisface la misma. La lógica utilizada para la especificación de los agentes es una lógica modal del conocimiento. Un agente se especifica en base a dos componentes: percepción y acción.

Pattie Maes [Mae89], [Mae90a], desarrolló una arquitectura de agentes, denominada *Agent Network Architecture* [Mae91]. Dicha arquitectura está compuesta por un conjunto de agentes, que consisten en una serie de módulos que se activan de forma independiente. Cada uno de los módulos se enlazan entre sí a través de sus pre-condiciones y post-condiciones, de forma análoga a los enlaces de una red neuronal.

Ferber [FD92] propone una arquitectura reactiva multiagente de **tareas competitivas**, en la cual cada agente debe decidir qué tarea debe realizar de entre varias posibles, seleccionando aquélla que proporciona el mayor nivel de activación. El sistema MANTA [DF92], [DCF95], [Fer96] utiliza esta misma aproximación, modelizando el sistema como una colonia de hormigas, en la cual cada hormiga decide qué acción debe realizar a fin de cumplir sus objetivos.

3.6.3. Arquitecturas híbridas

Una arquitecturas híbrida, como su nombre indica, combina características deliberativas y reactivas, utilizando tanto módulos deliberativos como reactivos, que colaboran dentro del agente. Los módulos reactivos llevan a cabo el procesamiento de aquellos estímulos que no precisan de deliberación. Por su parte, los módulos deliberativos establecen las acciones necesarias para satisfacer los objetivos locales y globales de los agentes. Generalmente, el módulo reactivo tiene prioridad en aquellas situaciones que requieren una reacción en un tiempo limitado.

Georgeff y Lansky [GL87] desarrollaron una de las más conocidas arquitecturas, PRS (*Procedural Reasoning System*). PRS es una arquitectura BDI, compuesta por una librería de planes y por representaciones simbólicas explícitas de las creencias, deseos e intenciones. No obstante, la librería de

planes contiene planes parcialmente elaborados, que se denominan áreas de conocimiento o KA (*knowledge area*). Las KAs pueden ser activadas por las metas o por los datos. Además, también pueden ser reactivas, permitiendo que el sistema PRS pueda responder rápidamente a cambios en el entorno. El sistema PRS se ha utilizado en simulación de entornos espaciales, así como en otros dominios [GI89b], [GI89a].

Ferguson desarrolló la arquitectura híbrida TOURINGMACHINES en su tesis doctoral [Fer92a], [Fer92b]. La arquitectura consiste en dos subsistemas, uno de percepción y otro de acción, que sirven de interfaz con el entorno de agente, y tres capas de control, que comunican ambos subsistemas [Fer95a]. La primera capa, la capa reactiva, permite responder de una forma rápida a los eventos. La segunda capa, la capa de planificación, construye planes y selecciona las acciones que se deben ejecutar para satisfacer las metas de los agentes [Fer95b]. La tercera capa, la capa de modelización, contiene representación simbólica del estado cognitivo de otras entidades en el entorno del agente. Estos modelos se manipulan a fin de resolver conflictos en las metas.

INTERRAP [MP94], [Mül94], [MPT94], [MPT95], [Mül96] es una arquitectura híbrida y estructurada en capas, al igual que TOURINGMACHINES, pero a diferencia de ésta, las capas son subdivididas en capas verticales, de modo que cada capa contiene una base de conocimiento y un conjunto de componentes de control que utilizan la misma. La capa más baja se llama interfaz con el mundo (*world interface*), que accede a su correspondiente base de conocimiento del modelo del mundo. Esta capa gestiona la interfaz entre el agente y su entorno. La capa siguiente es el componente basado en el comportamiento. El objetivo de este componente es la implementación y control de la capacidad reactiva del agente, a través del uso de patrones de comportamiento, PoB (*patterns of behaviour*). La siguiente capa está constituida por el componente basado en planes, que está formado por un planificador que genera planes para un único agente, en respuesta a las peticiones realizadas por el componente basado en comportamiento. La capa de mayor nivel es el componente de cooperación, que genera planes conjuntos para varios agentes, en respuesta a las peticiones realizadas por el componente basado en planes. Algunas aplicaciones se han basado en esta arquitectura [FMP96].

3.7. Comunicación de agentes

El objetivo fundamental de la cooperación entre agentes es la interacción entre los mismos para conseguir un objetivo común en un espacio distribuido. Para permitir la coordinación y cooperación entre agentes es necesario utilizar un marco donde puedan comunicarse. Se puede diferenciar un amplio rango de formas de comunicación [Wer89], que vamos a analizar a continuación.

3.7.1. Sin comunicación

Es posible tener varios agentes en un sistema y no utilizar ninguna forma de comunicación entre ellos (*communication-free*). La interacción en este caso se debe a la inferencia de las intenciones del resto de los agentes [RG88], [GGR88]. En muchas ocasiones, se utiliza la probabilidad como herramienta para la representación de la incertidumbre acerca de los movimientos del resto de los agentes [RB89].

3.7.2. Comunicación primitiva

El nivel más bajo de comunicación entre agentes lo constituye el trasiego de un número fijo de mensajes entre los agentes que forman el sistema. Así, en [Geo89] se utiliza un agente intermedio que se encarga de sincronizar a dos agentes que compiten por un determinado recurso. La comunicación

lograda mediante este esquema está muy limitada.

3.7.3. Comunicación mediante pizarra

Una forma de comunicación muy utilizada en las aplicaciones relacionadas con la Inteligencia Artificial es la denominada "arquitectura de pizarra" [Cor91], [CL92]. Para utilizar este método de comunicación, se utilizan tres elementos:

- Una pizarra, que constituye la base de datos global y que es el lugar donde se comparte el conocimiento del sistema.
- Varias fuentes de conocimiento, que intercambian el conocimiento que poseen a través de la pizarra.
- Un mecanismo de control, que permite controlar el acceso concurrente a la pizarra por parte de las fuentes de conocimiento.

Algunos ejemplos de arquitecturas de agentes que usan la comunicación mediante pizarra son [CC94], [CM01], [Wit92] y [CN96].

3.7.4. Paso de mensajes

Este paradigma constituye un medio de comunicación no exclusivo de agentes [Ari90], [And00]. No obstante, los agentes pueden utilizarlo a fin de comunicarse entre ellos. Para ello, es necesario definir el formato de los mensajes y el protocolo utilizado para el envío e interpretación de los mismos. Existen una gran cantidad de sistemas multiagente que utilizan este mecanismo de comunicación, tales como Accord [GS94] o MadKit [GF01].

3.7.5. Comunicación de alto nivel

Los agentes han constituido un paso evolutivo en el desarrollo de aplicaciones, permitiendo utilizar técnicas de inteligencia artificial en dicho campo. En ese sentido, se han estudiado las interacciones entre agentes en el nivel de conocimiento [New82], llegando a modelos como el BDI [RG91b], [RG95], [Rao96], que se describió anteriormente.

Para estudiar las interacciones entre agentes, se han aplicado a sistemas multiagente técnicas del campo del lenguaje natural, tales como [GS86] y [FA93].

Los lenguajes de comunicación actualmente utilizados, tales como KQML [CFF+92], [ARP93], [LF97a], [FLM97], pueden clasificarse en esta categoría (véase sección 3.11).

3.8. Cooperación entre agentes

La coordinación se define como la interacción entre un conjunto de agentes para llevar a cabo una determinada tarea [DLC89], [Gas92]. Por otro lado, la cooperación es una clase específica de coordinación, en la cual los agentes cooperan entre sí para detectar inconsistencias entre los resultados parciales de cada uno de ellos e integrar estos resultados con los suyos propios [HS95].

Las dos principales ramas para el diseño de métodos y arquitecturas dedicadas específicamente al problema de la cooperación entre agentes son [DR94]:

- Sistemas multiagente (MAS: *MultiAgent System*).
- Resolución de problemas distribuidos (DPS: *Distributed Problem Solving*).

La diferencia principal entre ambas disciplinas radica en la coordinación entre los agentes que constituyen el sistema. En el primer caso, los agentes tienen mayor autonomía y pueden tener objetivos globales diferentes. Por el contrario, en la DPS, existe un plan central explícito para la resolución del problema global.

Sólo analizaremos la primera rama, que es la que está más estrechamente relacionada con el tema de la agencia, aunque ambos campos se solapan y se realimentan.

3.9. Sistemas multiagente

De nuevo hemos de preguntarnos, ¿qué es un sistema multiagente o MAS? Al igual que el caso de los agentes también se han propuesto diferentes definiciones [FM99]. Desde el punto de vista de la Inteligencia Artificial, un MAS se define como una red de agentes, que trabajan en cooperación para encontrar respuesta a problemas que no se pueden resolver de forma individual por cada uno de ellos [DLC89].

Desde un punto de vista más general, en [JSW98] se define un MAS como un sistema compuesto por múltiples componentes autónomos que poseen las siguientes propiedades:

- Cada agente tiene la capacidad de solucionar el problema de forma parcial y no el problema global.
- No existe un sistema de control centralizado.
- Los datos no están centralizados.
- La ejecución es asíncrona.

Independientemente de cómo se defina un MAS, una de las características de los mismos es su modularidad, que permite que un problema complejo pueda ser resuelto a través de la participación de un conjunto de agentes, que se encargan de resolver aspectos particulares de dicho problema [JFL$^+$01].

El motivo del creciente interés que está suscitando el campo de los sistemas multiagente viene dado principalmente por la capacidad que tienen de realizar las siguientes tareas [Syc98]:

1. Resolver problemas demasiado complejos para un único agente centralizado.
2. Permitir la interconexión de múltiples sistemas heredados (*legacy systems*) a través, por ejemplo, de *wrappers* o envoltorios, que permitan la interconexión [GK94].
3. Proporcionar soluciones a problemas que pueden ser resueltos a través de una sociedad de componentes autónomos que interaccionan [GS96], [DBM$^+$92], [KLR$^+$92], [CMS83].
4. Proporcionar soluciones que utilicen de forma eficiente fuentes de información distribuidas espacialmente.
5. Proporcionar soluciones a entornos donde el conocimiento experto está distribuido [LS93].
6. Incrementar el rendimiento de las aplicaciones, aumentando la eficiencia, fiabilidad, extensibilidad, robustez, mantenimiento, rapidez, flexibilidad y reutilización de las mismas.

A pesar de que los MAS proporcionan una gran cantidad de ventajas, también presentan algunas dificultades [BG88c], [Syc98]:

1. La formulación, descripción y descomposición de problemas así como la obtención de resultados por parte de un grupo de agentes.

2. La comunicación e interacción de agentes.
3. La toma de decisiones por parte de los agentes a fin de obtener soluciones globales.
4. La representación y razonamiento de los agentes sobre las acciones, planes y conocimiento de otros agentes con los cuales se coordinan.
5. El tratamiento y resolución de los conflictos entre los agentes que se coordinan.
6. El diseño y metodología de desarrollo de MAS.

Tal vez el mayor reto de un MAS sea que exhiba un comportamiento colectivo coherente. La **coherencia** es una propiedad global del MAS que permite que el sistema se comporte como una unidad y que debe ser medida por la eficiencia, calidad y consistencia del comportamiento global del sistema. Existen diferentes métodos que permiten incrementar la coherencia y que se analizan a continuación.

3.9.1. Razonamiento individual sofisticado

El razonamiento individual sofisticado de cada agente puede incrementar la coherencia del MAS porque cada agente individual puede razonar sobre efectos no locales correspondientes a acciones locales, intuir el comportamiento de otros agentes o explicar y posiblemente reparar conflictos. Como ejemplo, en la infraestructura multiagente RETSINA [SPvVG96], [SPvVG01] cada agente tiene una arquitectura de razonamiento sofisticada que consiste en diferentes módulos que operan de forma asíncrona. Entre estos módulos se encuentra el módulo de comunicación y coordinación que se encarga del envío y recepción de mensajes, así como su interpretación.

3.9.2. Organizaciones

Una organización o estructura organizativa se define como un patrón de las relaciones de control e información entre los nodos del MAS. Una organización proporciona un entorno para la interacción de agentes a través de la definición de papeles, intuiciones sobre el comportamiento y relaciones de autoridad. En entornos abiertos, los agentes del sistema no están predefinidos estáticamente, sino que pueden formar parte del mismo de forma dinámica. En este escenario, se utilizan *agentes intermediarios* para la localización de los agentes. Cuando un agente es iniciado, anuncia sus capacidades al agente intermediario. Cualquier agente que busque un determinado agente con unas capacidades determinadas consulta al agente intermediario. Se han identificado diferentes clases de agentes intermediarios [DSW97].

Existen distintos tipos de estructuras organizativas:

- **Organización jerárquica:** Un único agente resolutor o un grupo especializado se encarga de realizar la toma de decisiones en cada nivel de la jerarquía. Los agentes superiores de dicha jerarquía ejercen el control sobre los recursos y las tomas de decisiones.
- **Comunidad experta:** Esta organización es plana y cada resolutor se especializa en un área particular del problema. Los agentes se relacionan utilizando determinadas reglas [LS93], [LLC91].
- **Organización de mercado:** El control se realiza de forma análoga al mercado, haciendo que los nodos compitan por las tareas y recursos realizando ofertas y contratos y evaluando las propuestas a través de la valoración económica de los servicios y la demanda [MW96], [SD83], [San93].

- **Comunidad científica:** Se toma como modelo la comunidad científica [KH81]. Las soluciones se construyen de forma local, se comunican a otros resolutores que pueden comprobar dichas soluciones y refinarlas [Les91].

3.9.3. Asignación de tareas

Este problema consiste en asignar responsabilidades y recursos para la resolución de tareas a un agente. Minimizar las interdependencias entre las tareas tiene dos ventajas respecto a la coherencia:

1. Mejora la eficiencia de la resolución de los problemas, ya que se decrementa la sobrecarga de comunicación existente entre los diferentes agentes.

2. Muy probablemente se mejore la consistencia de la solución, ya que se minimizan los conflictos potenciales.

La asignación de tareas puede ser resuelta mediante el protocolo CNP, que se analiza más adelante en este capítulo.

3.9.4. Planificación multiagente

La planificación multiagente [DLC89], [Dur96] tiene como objetivo hacer posible la cooperación entre los agentes mediante la realización de un plan que detalle las interacciones entre dichos agentes y sus acciones. Un plan multiagente intenta garantizar la consistencia global.

Los primeros trabajos realizados en este terreno corresponden a [CMS83] y [DLC89], donde las interacciones entre los agentes son guiadas por estrategias de cooperación que permiten mejorar el rendimiento colectivo.

Dependiendo de si el plan es creado por un solo agente o por todos los agentes, se distinguen dos tipos de planificación, a saber:

- *planificación centralizada:* un agente toma el papel de coordinador y se encarga de generar un plan global a partir de los planes de los diferentes agentes [SCHR$^+$88], [Geo89].

- *planificación distribuida:* cada agente genera planes parciales. Estos agentes pueden llevar a cabo negociaciones a fin de construir planes más coordinados. El algoritmo PGP (*Partial Global Planning*) [DL88], [DL89], permite la construcción de planes globales parciales a partir de los planes locales de diferentes agentes.

3.9.5. Cooperación funcionalmente precisa

Este modelo (*Functionally Accurate, Cooperative Distributed Systems*) [Les91] se caracteriza por el hecho de que los agentes no necesitan tener toda la información de forma local para resolver sus subproblemas, sino que para resolver el problema, deben interaccionar con el resto e intercambiar resultados parciales.

En contraposición a este modelo, se encuentra el modelo CA/NA (*Completely Accurate, Nearly Autonomous*), en el cual cada agente tiene toda la información necesaria para resolver el problema y si no es así, la solicita a otro agente.

Algunos sistemas que siguen este modelo son el sistema de pizarra Hearsay II [FL77], [LE80] y el entorno para supervisión de vehículos distribuido DVMT (*Distributed Vehicle Monitoring Testbed*) [DLC87], [LCD87].

3.9.6. Reconocimiento y resolución de conflictos

Debido a que los sistemas multiagente carecen de puntos de vista, conocimiento o control globales, existen grandes posibilidades de que surjan inconsistencias en las metas, planes o creencias de los agentes. Para lograr una resolución coherente del problema, estas discrepancias deben reconocerse y eliminarse.

Algunos métodos empleados para llevar a cabo este proceso son [AG92], [BG88b]:

- Una primera solución consiste en utilizar un agente que pueda conocer los estados de todos los agentes y determinar las diferencias y resolverlas. El problema de esta solución reside en el hecho de que este agente se convierte en cuello de botella del sistema y punto único de fallo.
- Se puede recurrir a un árbitro o criterio jerárquico para resolver discrepancias entre los distintos agentes [SCHR+88].
- Se pueden resolver conflictos utilizando casos similares del pasado [Syc89].
- Si las discrepancias son originadas por restricciones conflictivas, el problema puede resolverse si se relajan las restricciones que no sean esenciales [SF89].
- La técnica principal para resolver conflictos en un sistema multiagente es la negociación. En [SCHR+88] se propone utilizar la negociación de cara a resolver conflictos entre agentes.

3.9.7. Asignación de recursos

Otro aspecto crítico relacionado con los sistemas multiagente es la asignación de recursos de forma efectiva a los diferentes agentes. De nuevo, se puede utilizar la negociación en escenarios donde la escasez de recursos provoca que no se puedan cumplir todos los objetivos. Para ello se puede extender el protocolo CNP a fin de abarcar la negociación en la asignación de los recursos disponibles.

3.10. Compartición de planes multiagente

Un posible escenario de cooperación entre agentes es especificado en [HS95] y resuelto a través de la ejecución del siguiente conjunto de pasos:

- Se proporciona a los agentes metas, o lo que es lo mismo, descripciones del estado deseado del "mundo" o entorno.
- Los agentes realizan una serie de acciones.
- También construyen una serie de planes, con el fin de lograr la meta u objetivo.
- Deben tener planificada una serie de eventos.
- Deben ejecutar el plan de acuerdo a dicha planificación.
- La cooperación se logra debido a la ejecución de planes compartidos, así como a la planificación conjunta.

El agente tiene que tener el suficiente conocimiento para poder modificar el "mundo" o entorno, ejecutando las tareas pertinentes; además debe conocer qué agentes cooperantes pueden llevar a cabo dichas tareas. La actividad de un agente está formada por una serie de fases, que forman el denominado "bucle de agentes básico" (*basic agent loop*). Las fases correspondientes son:

- Fase de activación de metas: Una meta activa es similar a un deseo en un entorno BDI. En esta primera fase es necesario definir las metas que un agente debe intentar lograr mediante la ejecución de tareas. Un agente puede tener varias metas activas en un instante determinado, aunque debe haber una restricción: cada una de ellas debe ser compatible con las demás.

- Fase de construcción de planes: En esta fase, es necesario construir el plan que permita lograr las metas activadas en la fase anterior. Un plan es un conjunto de eventos, cada uno de los cuales se define por una tarea asociada, el agente ejecutor y un conjunto de restricciones sobre los recursos necesarios para llevar a cabo dicha tarea. Las dependencias se reflejan en un orden parcial de eventos, que determinan el orden cronológico de ejecución. Un agente puede usar una variedad de métodos para encontrar un plan adecuado. Entre las diferentes alternativas, se encuentran las siguientes:

  - Acceder a los planes a través de una biblioteca de planes, construida previamente. Es una solución muy poco flexible y su eficiencia depende del tipo de problema que se desee resolver.
  - Razonamiento hacia atrás desde las metas.
  - Razonamiento hacia adelante desde el estado actual.

- Fase de planificación: En esta fase es necesario escoger un plan óptimo (plan que tenga el menor coste, de acuerdo a una función de coste escogida apropiadamente) del conjunto de planes hipotéticos. Las tareas correspondientes se planifican de acuerdo al orden de eventos del sistema. Un plan es similar a la intención en el modelo BDI.

- Fase de ejecución: Esta fase consiste en la ejecución de la planificación obtenida en la fase anterior. Para llevarlo a cabo, la fase debe llamar a las tareas apropiadas y manejar las interrupciones de acuerdo a los eventos.

En este escenario, la cooperación entre diferentes agentes se lleva a cabo principalmente a través de la compartición de planes multiagente. Estos planes son similares a los planes de un único agente en el hecho de que tratan con interdependencias entre acciones. No obstante, su diferencia fundamental es que los primeros son más elaborados, ya que las acciones deben ser asignadas a diferentes agentes, los cuales se comunican entre sí. Por tanto, la cooperación se reduce al proceso de distribución de metas, planes y tareas a través de diferentes agentes.

3.10.1. Aplicaciones de sistemas multiagentes

Se han desarrollado diferentes aplicaciones y arquitecturas multiagente, que pueden abarcan diferentes dominios.

Las primeras aplicaciones de MAS aparecieron a mediados de los años 80 y se han ido incrementando a lo largo de los años, extendiéndose el dominio de las mismas, que abarca desde el control de procesos hasta la gestión de información.

Una de las primeras aplicaciones MAS fue la aplicación DVMT (*Distributed Vehicle Monitoring*) [Dur87] [DL89], [Dur96].

En [Par87] se describe el sistema YAMS (*Yet Another Manufacturing System*), que aplica el protocolo CNP al control de manufacturas. YAMS utiliza un sistema multiagente, donde cada componente del proceso es representado por un agente. Otros sistemas desarrollados en la misma área incluyen aquéllos que permiten el diseño de productos manufacturados [DB96] y diseño colaborativo [CFE+93].

El sistema multiagente más conocido para control de procesos es ARCHON, una plataforma software que permite construir MAS y una metodología asociada para construir aplicaciones con dicha

plataforma [JCL95a]. ARCHON ha sido aplicado en varias aplicaciones de control, tales como gestión de transporte, electricidad o control de un acelerador de partículas.

Otros sistemas de control de procesos basados en agentes han sido implementados para monitorizar y diagnosticar fallos en plantas nucleares [WW97], control aeronaútico [SQ93] y control climático [HC95]. OASIS [LL92] es un sistema de control de tráfico aéreo, en el cual los agentes permiten representar los sistemas de control aéreo.

Además, existen una variedad de aplicaciones MAS en el campo de las telecomunicaciones. En [WV94], se utilizan agentes que negocian la configuración de una llamada.

Flores y Wijngaards describen una arquitectura genérica MAS para colaboración dinámica en un entorno abierto [FW99]. Este trabajo considera el agente como una entidad social y enfatiza la importancia del factor social en la interacción de los agentes.

Otros problemas que utilizan sistemas basados en agentes para su resolución incluyen control de redes, transmisión, gestión de servición y gestión de redes.

3.11. Lenguajes de agentes

Al igual que los humanos poseemos una gran cantidad de lenguas, que en cierta medida limitan nuestro entendimiento común y por ello buscamos un lenguaje común para comunicarnos (la mayoría de las veces empleando el inglés como vehículo de entendimiento), los agentes también necesitan comunicarse e intercambiar información a través de estructuras reconocibles e interpretables. Estas estructuras forman lo que se denomina lenguaje de agentes.

El campo de los lenguajes de agentes recoge dos áreas de trabajo, a saber:

- Lenguajes de construcción del software de agentes, de forma que la implementación de los agentes responda a su arquitectura y especificación. También incluye la forma de compilación y ejecución de los mismos.

  Los lenguajes de construcción del software de agentes se pueden dividir a su vez en otros dos tipos de lenguajes, dependiendo del aspecto del agente que permitan programar:

  - Lenguajes de programación de la estructura del agente, es decir, aquéllos que permiten programar las funcionalidades básicas de un agente. Los lenguajes empleados pueden ser tanto lenguajes de propósito general(Java, C, C++, Lisp, Prolog, entre otros) así como lenguajes específicos, tales como April [MC95] del proyecto IMAGINE [Hau94] o CUBL (*Concurrent Unit Based Language*) del proyecto DAISY [Pog95]. Una de las principales desventajas del uso de lenguajes de propósito general es que el diseño del agente debe hacerse partiendo de cero. No obstante, la metodología de orientación a objetos permite facilitar esta tarea.
  - Lenguajes de programación del comportamiento del agente, es decir, aquellos lenguajes que permiten definir el conocimiento de un agente, tanto conocimiento inicial (por ejemplo, modelo de entorno BDI), como primitivas de mantenimiento del conocimiento (reglas, planes o similar) o primitivas de obtención de objetivos (reglas, planes o similar). Se pueden distinguir cuatro tipos de lenguajes de programación del comportamiento del agente:
    - Lenguajes de descripción de agentes: los agentes "heredan" de una clase agente genérica, que permite la descripción básica del modelo de agente. Esta descripción es traducida a un lenguaje ejecutable. Ejemplos de este tipo de lenguajes lo constituyen MACE ADL [GBH88] o AgentSpeak [WRR95].

- Lenguajes de programación orientados a agentes: Se trata de los lenguajes del denominado paradigma de orientación a agentes o AOP (*Agent Oriented Programming*), definida por Shoham [Sho93], [Sho97]. Estos lenguajes permiten modelar los estados mentales de los agentes y las transiciones entre ellos. Ejemplos de lenguajes de este tipo son AGENT0 [Sho91], [TV91], PLACA [Tho94] y Agent-K [DE94].
- Lenguajes basados en reglas de producción: Algunos lenguajes de agentes utilizan reglas de producción para programar la base de conocimiento. Algunos ejemplos de este tipo de lenguajes son DYNACLIPS [CKL94], basada en CLIPS [Gia88], MAGSY [Fis93], basada en OPS5 [For81] y RTA [WG94].
- Lenguajes de especificación: Son lenguajes que permiten especificar agentes. A partir de esta especificación se puede generar la conducta del agente, así como verificar sus propiedades. Ejemplos de estos lenguajes son METATEM [Fis94] y DESIRE [BDKJT97].

- Lenguajes de comunicación de agentes o ACL (*Agent Communication Language*), es decir, qué lenguajes deben emplear los agentes para comunicarse en su entorno. Los agentes interactúan y cooperan entre sí intercambiando mensajes y éstos pueden provenir del entorno (a través de sus sensores) o proceder de modificadores del entorno a través de sus operadores. Ambos sentidos deben poseer un determinado lenguaje para su compresión.

  Los lenguajes de comunicación de agentes pueden ser clasificados de la siguiente forma:

  - Lenguajes procedimentales: Utilizan directivas procedimentales, de forma que cuando un agente recibe un mensaje ejecuta un determinado procedimiento. Este tipo de lenguajes suelen basarse en lenguajes de *scripts*, tales como Perl o Tcl y se suelen utilizar en la implementación de agentes de usuario o agentes móviles. Sodabot [Coe94], Agent Tcl [Gra95], [Gra96] o Telescript [Whi95] constituyen ejemplos de este tipo de lenguajes.
  - Lenguajes declarativos: Utilizan sentencias declarativas para llevar a cabo las peticiones y las respuestas que forman parte de un acto de comunicación. También se pueden utilizar para definir otros aspectos relacionados con los agentes. El ejemplo más representativo de este tipo de lenguajes lo constituye KQML [FLM97], que forma parte del proyecto KSE (*Knowledge Sharing Effort*).

En la figura 3.1 se puede ver la clasificación de los diferentes lenguajes de agentes y además cómo encajan los diferentes protocolos en el cuadro general.

Dentro de los lenguajes vistos, el resto de este capítulo se va a centrar en la descripción de los lenguajes de comunicación de agentes, debido a que son ampliamente utilizados en sistemas multiagente.

Los agentes para su comunicación e interoperabilidad necesitan:

1. Un lenguaje común.
2. Unas ideas comunes sobre el conocimiento que intercambian.
3. La capacidad para poder intercambiar información que viene dada por los dos puntos anteriores.

Para resolver un problema tan complejo como el intercambio de información, las diferentes líneas de investigación comparten una sistemática común: la descomposición del problema en problemas más pequeños.

Dentro de estas líneas de investigación se encuentra una de las líneas más avanzadas tanto por sus novedades como por sus resultados, el KSE. El método empleado por el consorcio KSE considera que la capacidad de comunicación y compartición de información es de vital importancia, estableciendo los componentes necesarios de los agentes:

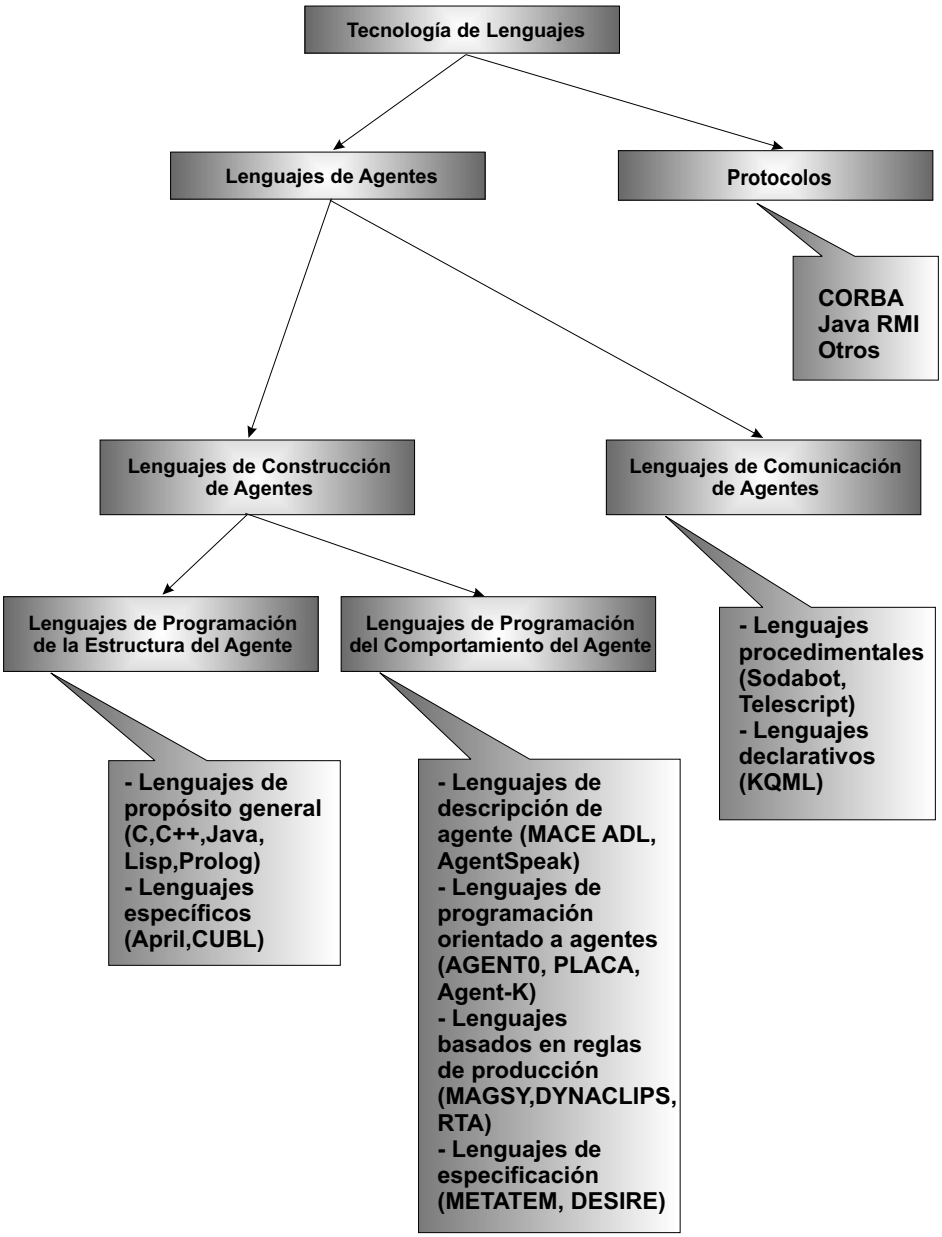


Figura 3.1: Taxonomía de los diferentes lenguajes de agentes

- Componentes de comunicación.
- Componentes de representación.
- Otros componentes o aquéllos no relacionados directamente con la compartición de información y conocimiento.

Asimismo, dentro del problema general, el traspaso de conocimiento mutuo está dividido en dos problemas:

- Traducción de los lenguajes de información.

- La compatibilidad o compartición del contenido semántico de la representación del conocimiento entre las diferentes aplicaciones.

Se contempla, por tanto, no sólo el problema de traducción de los sistemas de representación de conocimiento, sino también el problema de interpretación y significación de la representación.

Por otra parte, para resolver los aspectos relacionados con la comunicación, hemos de definir tres componentes claramente diferenciados:

- Protocolo de interacción.
- Lenguaje de comunicación.
- Protocolo de transporte.

Se considera protocolo de interacción al de mayor nivel estratégico y que permite establecer el conjunto de interacciones entre agentes. Estos protocolos pueden abarcar desde esquemas sencillos basados en axiomas hasta esquemas de negociación, pasando por protocolos de teoría de juegos. Un ejemplo de esquema de negociación lo constituye el protocolo de contratos o CNP, que ya hemos mencionado anteriormente y que a continuación se describe.

Se llama lenguaje de comunicación al medio a través del cual se diferencia el contenido de la actitud de la comunicación, pudiendo distinguir aserciones, respuestas o interrogaciones.

El protocolo de transporte es el mecanismo que permite llevar a cabo el "transporte" utilizado para la comunicación. Ejemplos de protocolos de transporte lo constituyen protocolos tales como SMTP o HTTP.

Sobre la base de estas premisas, el KSE ha determinado en sus investigaciones tres importantes vías [ANS98],[ARP93],[Gru93] que han obtenido tres productos sobre los que realizar su trabajo, a saber:

- KIF.
- Ontolingua.
- KQML.

3.11.1. Protocolo de contratos o CNP

CNP (*Contract Net Protocol*) [Smi80] [SD81], [Smi88], es un protocolo muy utilizado en los sistemas multiagente, que permite llevar a cabo el control de tareas por parte de agentes cooperantes. Para ello, este protocolo se basa en la realización de contratos entre diferentes agentes. El agente que contrata a otros toma el papel de *gestor*, y los agentes que realicen la tarea toman el papel de agentes *contratados*. El gestor se encarga de supervisar a los agentes contratados. Para llevar a cabo la asociación entre gestor y contratados se lleva a cabo un proceso de negociación, que consiste en el envío de ofertas por parte del gestor a los posibles contratados. Éstos las evaluarán, aceptándolas o rechazándolas. En el caso de que la oferta se acepte, se afirma que se ha "firmado un contrato" por parte de ambos agentes. Los agentes contratados pueden a su vez actuar como gestores, subcontratando la tarea que deben realizar o parte de ella.

Sandholm y Lesser realizaron extensiones al protocolo CNP, debido a la necesidad de trabajar con la incertidumbre presente en los procesos de negociación [San93]. Otras extensiones fueron llevadas a cabo por Sycara, introduciendo teoría financiera a fin de obtener esquemas contractuales flexibles en entornos variables [Syc97].

3.11.2. KIF (*Knowledge Interchange Format*)

KIF es la solución que presenta el consorcio KSE para los aspectos sintácticos de la representación del conocimiento. Pretenden con este lenguaje tener una poderosa herramienta para expresar lenguaje y metalenguaje. Y se formuló con dos directrices diferentes:

1. La creación de un lenguaje para el desarrollo de la Inteligencia Artificial dirigido a la interoperatividad entre componentes.

2. La creación de un formato común de intercambio que permitiera interacción con otros lenguajes.

3.11.3. Ontolingua

Una **ontología** define un conjunto de clases, funciones, y objetos constantes propios de cada dominio del discurso e incluye una axiomatización para construir la interpretación. El resultado permite que las sentencias sean interpretadas sin ambigüedades e independientes del contexto y dejando explícitos los detalles relevantes. Las ontologías son de gran utilidad en las aplicaciones de comunicación. De hecho, se establece que "un agente representa una ontología si sus acciones observables son consistentes con la definiciones de la ontología".

Ontolingua es una herramienta que permite intercambiar ontologías entre diferentes entidades.

Los trabajos sobre el problema de contenido del conocimiento representado de manera formal, llevaron a la utilización de un formalismo KIF, y de un lenguaje de comunicación que se describe a continuación, KQML. Para ello, KSE realiza la construcción de ontologías en varios dominios. Se escriben éstas en KIF, usando un vocabulario de definiciones de ontología.

3.11.4. KQML (*Knowledge Query Manipulation Language*)

KQML ha sido concebido como un formato de mensaje y un protocolo de manejo de mensajes para su uso entre agentes que comparten su conocimiento en tiempo real. Los mensajes KQML son transparentes u opacos, según como se entienda, al contenido que transportan. Éstos comunican sentencias en cualquier lenguaje, pero enviando además una actitud respecto a dicho contenido (aserción, respuesta, pregunta). Las primitivas del lenguaje se llaman *performatives* (directivas), que es un término perteneciente a la teoría de lenguajes. Dichas directivas definen las acciones permitidas u operaciones que los agentes pueden esperar que ocurran en la comunicación con otros agentes.

Hacer una distinción entre protocolo y lenguaje de comunicación es, según los expertos, a menudo difuso.

Un protocolo de comunicaciones puede encajar con una de estas tres clases:

- Un protocolo de transporte tal y como se han definido anteriormente (HTTP, SMTP o FTP entre otros).

- Una aplicación marco de alto nivel para interacción, como negociación, protocolos de teoría de juegos, planificación, etc.

- Un posible intercambio válido de primitivas de comunicación.

Dar un lenguaje común a agentes que permita aminorar las dificultades de comunicación consiste en generar una sintaxis, una semántica y una norma práctica o praxis.

El concepto de representación y recuperación de información no es universal y existen múltiples lenguajes diferentes que intentan resolver el problema que se presenta, tal como SQL. Ante la existencia de muchos lenguajes que se pueden encontrar en su entorno, la solución más óptima parece ser el uso de

lenguajes de traducción entre los distintos lenguajes nativos. Si se acepta la viabilidad de la traducción, todavía queda el problema de conocimiento del contexto que deben compartir con el fin de interpretar el contenido del mensaje no sólo en el plano semántico, sino también en el ontológico.

Los contenidos de las ontologías en ciertos campos como medicina, biología o bolsa constituyen un aspecto importante de la comunicación.

Por ello, KQML se configura en primer lugar según las normas prácticas o praxis y en segundo lugar sobre la base de la semántica. Se obtiene así un lenguaje y un conjunto de protocolos que permiten conectar e intercambiar información entre agentes.

KQML se divide en tres niveles:

- Un nivel de contenido.
- Un nivel de mensaje.
- Un nivel de comunicación.

El nivel de contenido alberga el contenido real del mensaje, que puede incluir varios tipos de codificación expresables en notación binaria, texto o ASCII, y otras que puedan contener información intercambiable. Para lograrlo, la implementación de KQML ignora la parte del mensaje en sí, interesándose únicamente por su longitud (dónde comienza y dónde termina).

En el nivel de comunicación se engloba un conjunto de características que se tratan a bajo nivel como parámetros relacionados con la comunicación, como son la identidad del remitente y del receptor y el identificador único asociado a la comunicación.

El nivel de mensaje se usa para codificar un mensaje que una aplicación puede mandar a otra. Esta capa forma parte del núcleo básico del lenguaje KQML y determina las formas de interacción de los agentes que utilicen KQML.

La sintaxis de KQML se basa en listas limitadas por paréntesis, incapaz de ocultar las raíces del lenguaje, es decir el lenguaje Common LISP. El elemento inicial de la lista es una directiva; los restantes argumentos son pares de nombres clave y valores.

Ejemplos de mensajes de un agente a otro son:

```
(ask-one
:sender agente_remitente
:content (PRICE aralia >price)
:receiver bolsa-madrid
:reply-with aralia_accion
:language LPROLOG
:ontology MSE-TICKS)
```

En este mensaje se utiliza la directiva para preguntar por el valor de las acciones de una empresa ficticia denominada Aralia en la Bolsa de Madrid, que debe responderse mediante el lenguaje Prolog y con la ontología propia de las cotizaciones de la Bolsa de Madrid.

```
(tell
:sender bolsa-madrid
:content (PRICE aralia 15,36)
:receiver agente_remitente
:in-reply-to aralia_accion
:language LPROLOG
:ontology MSE-TICKS)
```

Este mensaje podría ser una respuesta al anterior, en la que se devuelve el valor de la acción dada la ontología. No es necesario poner la moneda, ya que se entiende que la cantidad viene dada en euros (esto debe ser descrito por la ontología).

Uno de los requisitos del lenguaje KQML es que pueda interactuar con una amplia variedad de arquitectura de agentes. El método de introducir un pequeño número de directivas mediante las cuales los agentes pudieran descubrir las especificaciones correspondientes a la metainformación, con sus requisitos y capacidades, permite definir una clase especial de agentes llamados "facilitadores" (*facilitators*). Un facilitador es un agente intermedio que realiza varios servicios, entre los que se encuentra el mantenimiento del servicio de nombres, es decir, asociación de nombres simbólicos con direcciones físicas, servicios de enrutamiento de mensajes según contenido, reconocimiento de patrones entre informaciones de proveedores y clientes o servicios de medición y traducción. Por tanto, los facilitadores son agentes que trabajan utilizando conocimiento sobre los servicios de información y que son capaces de solicitar a otros agentes dicha información y además ofrecer servicios de *forwarding* o *brokering*.

3.11.5. Un lenguaje multi-agente: MAI2L

MAI2L [HS95] es un caso práctico de lenguaje de comunicación, más restrictivo que KQML y con un entorno de aplicación más cerrado. En resumen, se trata de una respuesta menos ambiciosa, aunque conceptualmente similar a KQML. El prototipo de este lenguaje está implementado en C y disponible en plataformas UNIX y Windows. La base de conocimiento (*KB: Knowledge Base*) de un agente en este sistema está formada por tuplas, denominadas términos MAI2L. Las acciones MAI2L y los planes MAI2L son términos especiales en este sistema. Hay cuatro aspectos a tener en cuenta respecto a las acciones y planes, a saber:

- **Personajes o participantes:** Representan aquellos agentes que participan en una acción o en un plan multiagente. Dicho plan se compila en planes separados para cada participante y contiene puntos de sincronización.
- **Precondiciones:** Son las condiciones que se deben satisfacer para que la acción o el plan se ejecute.
- **Procedimientos:** Consiste en una llamada a una función. La ejecución de la misma se lleva a cabo en el núcleo del agente.
- **Efectos:** El efecto de un plan o acción modifica la KB.

Debido a que MAI2L se ha diseñado como un lenguaje multi-agente, soporta distribución de metas, planes y tareas entre diferentes agentes. Para lograr dicho objetivo, utiliza primitivas de cooperación a nivel básico y métodos de cooperación a un mayor nivel. Cuando se crea un agente, se registra mediante el servicio de directorios ADS (*Agent Directory Service*), que es a su vez otro agente (en la terminología anteriormente citada, un facilitador). ADS almacena información sobre los detalles de comunicación de los agentes y sus tipos y capacidades. A continuación, la parte de comunicación del agente espera nuevas conexiones, ejecutando las acciones *send* y *receive* predefinidas, las cuales son planificadas por el núcleo del agente.

- La acción *receive* tiene como argumentos una lista de agentes del cual puede recibir mensajes el agente que hace la llamada, así como un *timeout* o tiempo de espera. La parte de comunicación envía el contenido del mensaje a cada agente, de forma que se lleve a cabo el procesamiento correspondiente.

- La acción *send* permite enviar un mensaje a un solo agente o a una lista de agentes. Esta acción debe determinar la dirección del agente o agentes (consultando al ADS si fuera necesario), generar los términos MAI2L correspondientes y finalmente llevar a cabo el envío.

Las primitivas de cooperación representan la transferencia de conocimiento de un agente a uno o más agentes mediante términos MAI2L, utilizando una meta específica. Por otro lado, los métodos de cooperación son planes multi-agente independientes del dominio que se utilizan para construir y ejecutar planes multi-agente específicos del dominio.

3.12. Agentes y sistemas de recuperación de información

Una de las posibles aplicaciones de los agentes es la gestión de información. A este tipo de agentes se les denomina agentes de información (*information agent*).

Papazoglou et al. [PLS92] definen un agente de información como un agente que tiene acceso a, al menos, una y potencialmente muchas fuentes de información y es capaz de recoger y manipular dicha información de cara a responder a las consultas realizadas por los usuarios y por otros agentes de información.

De acuerdo a la clasificación de Dick Stenmark [STE], los agentes de información se pueden clasificar en:

- Agentes interfaz: Se utilizan para decrementar la complejidad de sistemas de información más sofisticados y sobrecargados. Algunos de ellos permiten interpretar el lenguaje natural.
- Agentes de sistema: Ejecutan como parte integrante de los sistemas operativos o de dispositivos de la red. Realizan labores tales como inventario de hardware, interpretación de los eventos de red, gestión de copias de seguridad o de dispositivos de almacenamiento o detección de virus. Estos agentes no suelen trabajar de forma directa con las aplicaciones de usuario.
- Agentes asesores: Se emplean en complejos sistemas de ayuda o de diagnosis.
- Agentes de filtrado: Se utilizan para reducir la sobrecarga de información eliminando datos que no encajan con determinados perfiles.
- Agentes de recuperación de información (*information retrieval agents*): Se encargan de buscar y recuperar información y se pueden utilizar como gestores de documentos.
- Agentes de "navegación": "Navegan" a través de redes internas y externas, recordando enlaces, pre-cargando información y almacenando enlaces interesantes.
- Agentes de monitorización: Proporcionan al usuario información sobre determinados eventos que se encarga de monitorizar, tal como movimientos, actualizaciones o borrados de información.
- Agentes de recomendación: Este tipo de agentes se utiliza de forma colaborativa. Estos agentes realizan recomendaciones sobre determinadas decisiones a tomar por los sistemas.
- Agentes para *profiling*: Estos agentes construyen *sites* con información y recomendaciones adaptadas a las necesidades individuales de cada usuario.

Algunos sistemas utilizan agentes que cumplen varios de estos papeles.

Capítulo 4

E/S distribuida y paralela

4.1. Introducción

Salvo raras excepciones, las aplicaciones requieren interactuar con el exterior, a través de la entrada o salida de datos. El sistema de E/S es el encargado de gestionar en el computador los dispositivos que ejercen de interfaz del mismo. Aunque tradicionalmente se ha discriminado la importancia de este sistema frente al estudio del procesador, lo cierto es que constituye una parte fundamental de la arquitectura de cualquier computador. Como afirma Patterson en el prólogo de [JCB02], "Un computador sin dispositivos de E/S es como un coche sin ruedas - no puedes llegar muy lejos sin ellas".

La mayoría de las aplicaciones utilizan los ficheros como una forma de almacenamiento permanente de la información y como medio de compartición con otras aplicaciones. El sistema de ficheros es la parte del sistema de E/S que se encarga de la gestión de los ficheros y proporciona un mecanismo de abstracción a los usuarios, de forma que oculta todos aquellos detalles relacionados con el almacenamiento y distribución de la información en los discos, así como el funcionamiento de los mismos.

Los sistemas de ficheros tradicionales o secuenciales son aquellos sistemas que se ejecutan en entornos monoprocesador o SMP y ofrecen sus servicios únicamente a los usuarios de dichos sistemas. Un ejemplo de sistema de este tipo lo constituye el sistema de ficheros de UNIX (UFS: *UNIX File System*) [Bac86]. En UNIX cada fichero se representa como una secuencia de bytes lineal y un puntero a partir del cual se realizan las operaciones de E/S [Tho78]. UFS proporciona un conjunto de operaciones básicas de acceso a ficheros, que permiten acceder de forma contigua a los datos almacenados en discos locales. Esta interfaz no es adecuada para los requisitos de aplicaciones distribuidas y paralelas.

4.2. Sistemas de ficheros distribuidos

Con la aparición de las redes de interconexión, surgió la necesidad de compartir datos entre diferentes máquinas, problema que originó la aparición de los sistemas de ficheros distribuidos [Svo84],

[LS90]. Los sistemas de ficheros distribuidos permiten que procesos de múltiples computadores puedan acceder a un conjunto común de ficheros. Un sistema de este tipo garantiza la transparencia de acceso a los ficheros en un entorno distribuido, ofreciendo un espacio de almacenamiento global que permite a múltiples clientes compartir los dispositivos de almacenamiento.

Aunque un sistema de ficheros distribuido está formado por múltiples servidores y dispositivos de almacenamiento, el acceso a la información no se realiza en paralelo.

NFS [SGK+85], AFS [HKM+88], Coda [SKK+90], Sprite [OCD+89] o GFS [SRO96] son ejemplos de sistemas de ficheros distribuidos.

4.2.1. NFS

NFS (*Network File System*) fue diseñado originalmente por Sun Microsystems en 1985 [SGK+85]. NFS permite el acceso transparente a ficheros y directorios situados en máquinas remotas, para poder utilizarlos como si fueran locales. Sigue un esquema cliente-servidor, de forma que el nodo remoto se denomina servidor NFS y el nodo local se denomina cliente NFS. El servidor sitúa directorios de su propio árbol de directorios a disposición de otros nodos que se encuentran conectados a través de una red. Para llevar a cabo esta tarea, el servidor debe exportar dichos directorios. Los clientes a fin de utilizar los directorios exportados previamente deben "montarlos" en algún directorio de su propio sistema de ficheros, directorio al cual se denomina "punto de montaje".

Los clientes y servidores se comunican a través de la red mediante el uso de RPC. El servidor NFS tiene una interfaz que permite programar cualquier cliente para comunicarse con el mismo [NFS89], [NFS95].

El protocolo NFS fue diseñado **sin estado**, de modo que el servidor escribe de forma estable los datos que hayan sido modificados antes de devolver los resultados al cliente. El cliente hace peticiones al servidor que incluyen toda la información necesaria para completar la operación. Esta propiedad del servidor le permite que sea sencillo recuperarse de un fallo, ya que si un cliente "cae" no afecta a las operaciones del servidor o de otros clientes. Si es el servidor el que falla, éste sólo necesita reiniciarse.

El protocolo NFS proporciona otro protocolo denominado NLM (*Network Lock Manager*), que permite gestionar cerrojos sobre los ficheros.

NFS de Sun Microsystems se transformó en el éstándar para compartir archivos bajo plataforma UNIX. Actualmente existen implementaciones de NFS para otros sistemas operativos. Como ejemplo, Linux tiene dos implementaciones diferentes del servidor NFS: un servidor NFS en espacio de usuario y otro servidor NFS en el núcleo del sistema operativo [Lu99].

4.2.2. AFS

AFS (*Andrew File System*) es un sistema de ficheros distribuido que permite una cooperación entre diferentes nodos a fin de compartir de una forma eficiente los recursos a través de una LAN o una WAN. Para ello, AFS utiliza un esquema cliente-servidor

AFS fue desarrollado originalmente en el Centro de Información Tecnológica de la Universidad de Carnegie-Mellon.

La unidad organizativa más importante de AFS es la celda, que consiste en un conjunto de servidores agrupados, que almacenan la estructura de los volúmenes o subárboles en los que se divide el árbol de directorios completo AFS. Además también existen servidores de autenticación, que permiten gestionar la seguridad de acceso a AFS a través del protocolo Kerberos.

El uso de cache permite incrementar el rendimiento de AFS, reduciendo de forma significativa el tráfico por la red. AFS está disponible para muchas plataformas, entre las que se incluyen SUN, HP, SGI, DEC o IBM.

4.2.3. Coda

Coda (*COnstant Data Availability*) es un sistema de ficheros implementado en la Universidad Carnegie Mellon y descendiente del sistema de ficheros AFS. De hecho, fue diseñado con el objetivo de mejorar la disponibilidad de los datos existente en AFS. En Coda pueden existir múltiples copias de cada fichero en diferentes servidores. De este modo, proporciona tolerancia a fallos al uso de los ficheros, eliminando el impacto que supone que un servidor deje de funcionar e incluso permitiendo que los clientes puedan trabajar en modo desconectado, cuando tienen en memoria local los ficheros correspondientes.

4.2.4. Sprite

Sprite es un sistema operativo distribuido que fue desarrollado bajo el proyecto de investigación SPUR (*Symbolic Processing Using RISCs*). Al igual que NFS, Sprite utiliza RPC para llevar a cabo la comunicación a través de la red.

El sistema de ficheros de Sprite está también distribuido a través de múltiples servidores y clientes y proporciona transparencia de nombres a la vez que un buen rendimiento.

El servidor Sprite registra los accesos a los ficheros y gestiona la consistencia de caches de los diferentes clientes [NWO88]. No utiliza disco local y proporciona una semántica UNIX.

Las operaciones de lectura tienen un rendimiento excelente. Por el contrario, las operaciones de escritura tienen un rendimiento muy limitado, debido a las altas latencias asociadas a las escrituras individuales de bloques de datos al disco y las actualizaciones de metainformación asociadas. Sprite utiliza la técnica LFS (*Log-structured File Storage*) [RO92] para incrementar el rendimiento de estas operaciones de escritura, acumulando un conjunto de dichas operaciones en memoria y, posteriormente, escribiéndolas a disco en grandes y contiguos segmentos de datos de tamaño fijo. Estos segmentos se denominan *log segments*, debido a que se escriben en el orden exacto en el que se actualizaron. De este modo, se evita la sobreescritura de los datos, haciéndose necesario el uso de un sistema de recolección de basura o *garbage collector*, que se encargue de liberar segmentos que ya no se utilicen.

4.2.5. xFS

xFS fue propuesto por un grupo de UC Berkeley. xFS consiste en un sistema de ficheros "sin servidor" (*serverless network file system*) [ADN+96], lo que implica que distribuye las responsabilidades del sistema de ficheros, el almacenamiento y los recursos a través de los nodos correspondientes de la red de área local. xFS surge como respuesta a las limitaciones de los sistemas distribuidos tradicionales, aprovechando las oportunidades ofrecidas por redes de área local más eficientes, tales como ATM o Myrinet. xFS implementa un sistema de almacenamiento tipo RAID (véase sección 4.3.1), distribuyendo los datos a través de los discos de cada uno de los nodos. xFS utiliza también una técnica basada en *logs*.

Este tipo de sistemas de ficheros dan un buen soporte a SAN (véase sección 4.3.2) y proporcionan alta disponibilidad, al no basarse en un único servidor central, que constituya un único punto de fallo en el sistema. Algunos autores denominan a este tipo de sistemas de ficheros *Shared-Disk File Systems* o incluso *Cluster File Systems*, ya que los datos se distribuyen entre todos los dispositivos de los nodos de la red de área local. Otros sistemas de ficheros que encajan dentro de esta clasificación son Zebra [HO01], Frangipani [TML97], GFS y Petal [LT96].

4.2.6. GFS

GFS (*Global File System*) consiste en un sistema distribuido que permite a múltiples nodos acceder y compartir discos y dispositivos de cintas en una red de almacenamiento [SRO96]. Al igual que xFS, GFS constituye un sistema de ficheros "sin servidor".

La primera versión de GFS se conoció como GFS-1 y apareció en verano de 1995. En dicha versión, se utilizó tecnología Fibre Channel a fin de procesar grandes *datasets* científicos en equipos Silicon Graphics. GFS fue por tanto implementado en el sistema operativo IRIX de Silicon Graphics. Esta implementación utilizaba discos SCSI paralelos y mandatos para llevar a cabo la sincronización, los cuales bloqueaban completamente los dispositivos, haciendo por tanto imposible acceder de forma simultánea a la metainformación. Este cuello de botella fue resuelto en la segunda versión, denominada GFS-2, a través de la implementación de grano fino de los cerrojos, que no bloqueaban todo el dispositivo [Sol97]. Esta solución no era escalable, debido a la contención de la red. Además, se requería el uso de grandes ficheros para obtener un buen rendimiento, debido a que no había cache en los clientes ni para la metainformación ni para los datos de los ficheros.

En 1998 se empezó a portar el código al sistema operativo Linux, integrando GFS en el kernel. De esta forma, se logró construir un sistema de ficheros de propósito general escalable y que permite la compartición de dispositivos a través del uso de una red heterogénea. Además, permite realizar *caching* de metainformación y ficheros. Esta versión se denominó GFS-3 [PBB+99].

4.3. E/S paralela

El uso de las redes de interconexión introduce otro segundo reto, que consiste en la ejecución eficiente de todo tipo de aplicaciones, tanto distribuidas como paralelas. En este sentido, hay que destacar la amplia difusión de servidores Web y la tendencia actual en el campo de la supercomputación hacia la sustitución de los grandes y costosos supercomputadores por redes o clusters de estaciones de trabajo. Así lo refleja [TOP99], donde aparecen varios clusters de estaciones de trabajo entre los 500 supercomputadores más potentes del año.

Los sistemas de ficheros distribuidos ofrecen un espacio de almacenamiento global que permite a múltiples clientes compartir los dispositivos de almacenamiento. En estos sistemas cada fichero se almacena en un servidor, y el ancho de banda de acceso a un fichero se encuentra limitado por el acceso a un único servidor, lo que convierte a los servidores en un cuello de botella en el sistema. Este problema, que se ha dado en llamar **crisis de la E/S** [PGK88], [KGP89], [Pat94] sigue sin estar resuelto en sistemas distribuidos de propósito general.

Dicha crisis es originada por el desequilibrio existente entre el tiempo de cómputo y el tiempo de E/S, ya que esta diferencia supone un salto cualitativo en la escala de tiempos. Mientras que el rendimiento de los procesadores y de los sistemas de computación se ha incrementado de forma drástica en los últimos años (*ley de Moore*)[Moo65], [Moo75] la velocidad de acceso de los sistemas de almacenamiento no ha seguido la misma evolución[1]. Esta diferencia es aún más acusada en sistemas multiprocesadores, donde, en el caso ideal, se multiplica el rendimiento del procesador por el número de procesadores del sistema. Mientras este salto no se acorte, la E/S continuará siendo uno de los principales cuellos de botella de la denominada HPC (*High-Performance Parallel Computing*).

Se han propuesto diferentes soluciones para resolver el problema de la crisis de la E/S, entre las que cabe destacar las siguientes:

- Utilización de paralelismo en el sistema de E/S con la distribución de los datos de un fichero entre diferentes dispositivos y servidores.

[1] La capacidad de almacenamiento sí se ha visto incrementada en una tasa de un 60 a un 80 % por año

- Empleo de sistemas de almacenamiento de altas prestaciones (redes de almacenamiento).

4.3.1. Empleo de paralelismo en el sistema de E/S

La idea original de la E/S paralela procede de la aparición del sistema RAID (*Redundant Array of Inexpensive Disks*). Estos sistemas fueron descritos inicialmente a principios de los años 80 [Law81], [PB86], aunque se hicieron populares debido al trabajo realizado por un grupo de investigadores de UC Berkeley [PGK88], [KGP89], [PCGK89].

Un RAID consiste en un conjunto de dos o más discos que, dependiendo del nivel de RAID que se implemente, ofrecen mejor rendimiento y mayor grado de tolerancia a fallos que un único disco, debido a que distribuyen los datos a través del conjunto de discos y almacenan y utilizan, si fuera necesario, datos redundantes para recuperarse de fallos físicos de los dispositivos.

[PGK88] y [CGKP90] definen inicialmente seis organizaciones RAID diferentes:

- **RAID nivel 0:** Consiste en un *array* de discos no redundante y, por tanto, no proporciona tolerancia a fallos. Sólo permite la distribución de los datos.
- **RAID nivel 1:** Consiste en un *array* de discos espejos. Los datos son duplicados para incrementar su disponibilidad.
- **RAID nivel 2:** Se trata de un *array* de discos que utiliza el código de Hamming como código de redundancia.
- **RAID nivel 3:** *Array* de discos que utiliza como protección de los datos la paridad de los mismos y como unidad de distribución de los datos el byte.
- **RAID nivel 4:** Igual que RAID nivel 3, excepto que la unidad de distribución de los datos es el bloque.
- **RAID nivel 5:** Igual que RAID nivel 4, pero la paridad se distribuye a lo largo de los discos.

Los diferentes niveles RAID ofrecen grandes diferencias entre rendimiento y tolerancia a fallos.

De estos sistemas, los más empleados son el RAID nivel 1 (*mirroring*) [BG88a], [CHLY95], [CK89] y los sistemas basados en la paridad [GP93], [GHW90], [LK91], entre los que destaca RAID nivel 5. El sistema RAID nivel 1 tiene como principal ventaja la fácil recuperación de los datos en caso de fallo en uno de los discos, en detrimento del espacio de almacenamiento, del que hace un pobre aprovechamiento, ya que todos los datos son duplicados. Por otro lado, en el sistema RAID nivel 5 las operaciones de E/S tienen mejor rendimiento. Las operaciones de lectura son eficientes debido a la posibilidad de leer en paralelo. Las operaciones de escritura son más costosas, por el hecho de tener que generar la paridad. Por otro lado, la recuperación es costosa, pero el aprovechamiento del espacio de almacenamiento es mejor.

El sistema HPRaid [WGSS96] implementa una jerarquía de dos niveles: el primer nivel utiliza *mirroring* de los datos activos; el segundo nivel utiliza paridad RAID 5 para los datos no activos. El sistema se encarga de gestionar de forma automática y transparente la migración de bloques de datos entre los dos niveles, combinando las ventajas de rendimiento del RAID nivel 1 con la capacidad del sistema RAID nivel 5.

Muchos trabajos han sido publicados sobre los sistemas RAID, su rendimiento y variaciones de diseño para optimizar los mismos. Se puede encontrar bibliografía comentada sobre los sistemas RAID en [CLG+94].

Posteriormente han surgido otros niveles de RAID, tales como RAID-6, RAID-7, RAID-10 o RAID-53.

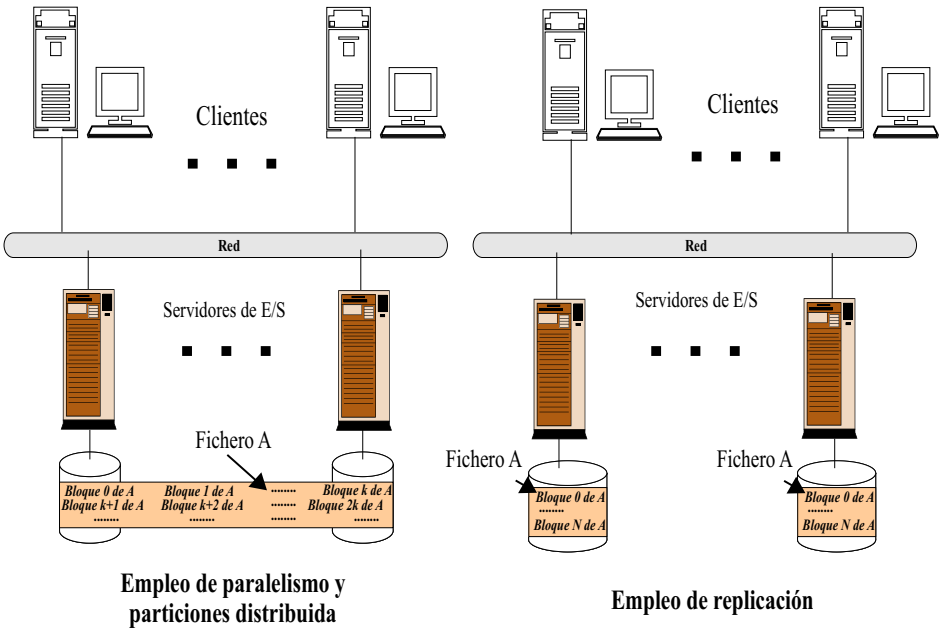


Figura 4.1: Diferencias entre replicación y paralelismo en el sistema de E/S

Si los sistemas RAID son utilizados por un servidor de ficheros tradicional, el ancho de banda de los discos es limitado por el ancho de banda de la memoria del servidor. Por el contrario, si varios servidores se utilizan paralelamente, el ancho de banda del sistema de E/S puede incrementarse de dos formas:

1. Se utilizan varios discos independientes, de forma que el sistema de ficheros puede acceder a los datos en paralelo, ya que éstos se distribuyen a través de los diferentes discos.

2. La distribución de datos mediante el empleo de particiones distribuidas [CPdM+97b], lo que permite que un único fichero se distribuya entre varios discos y servidores de E/S. Esta solución permite acceder en paralelo a los datos de un mismo fichero, algo que no se puede conseguir con el empleo de sistemas de ficheros distribuidos tradicionales. Hay que hacer notar, sin embargo, que el empleo de paralelismo en el sistema de ficheros es diferente al empleo de sistemas de ficheros replicados. En un sistema de ficheros replicado, cada disco (en cada servidor) almacena una copia completa de un fichero. Utilizando E/S paralela, cada disco (en cada servidor) almacena una porción del fichero, lo que permite acceder al fichero en paralelo utilizando menos discos y sin crear problemas de consistencia. La figura 4.1 muestra la diferencia entre estos dos enfoques.

Por tanto, el empleo de paralelismo en el sistema de E/S puede realizarse sobre un sistema distribuido compuesto por varias máquinas con discos conectados, lo que permite incrementar el ancho de banda del sistema de E/S [dRBC93] si se explota el acceso a los datos en paralelo. El paralelismo se consigue con la distribución de los datos entre varios servidores y discos, lo que permite leer (y escribir) los datos de los ficheros en paralelo desde diferentes servidores.

En la arquitectura genérica de un sistema de E/S distribuido-paralelo pueden distinguirse diferentes niveles funcionales [Eck90]:

- **Nodo de proceso (NP)**, realiza las peticiones de E/S. Se le denomina también cliente.

- **Nodo de E/S (NES)**, ejecuta las operaciones de E/S sobre ficheros y dispositivos lógicos. Se le denomina también servidor.
- **Controlador**, transfiere bloques físicos desde/hacia los dispositivos físicos.
- **Dispositivo**, almacena los datos en bloques físicos.

Actualmente existen dos alternativas para la implementación de paralelismo en el sistema de E/S:

- **Bibliotecas de E/S paralela.** La computación paralela se utiliza en diferentes tipos de aplicaciones, cada una de las cuales tiene requisitos muy diferentes. Por este motivo, han surgido diversas bibliotecas de E/S paralela que ofrecen a los programadores de aplicaciones un conjunto de funciones de E/S altamente especializadas y que intentan obtener el máximo rendimiento y flexibilidad para cada clase de aplicación.

  PASSION (*Parallel And Scalable Software for I/O*) [CTB+96], [CBH+94], [TBC+94] y Panda [SW94], [SCJ+95], [SW96] son dos ejemplos de bibliotecas de E/S con APIs especiales para la parte de E/S paralela.

  PASSION intenta dar solución al problema de la E/S proporcionando un lenguaje, un compilador y un entorno de ejecución. Este proyecto utiliza diferentes técnicas para optimizar los accesos no contiguos, entre las que se incluyen el cribado de datos y la técnica de dos fases (véase sección 4.3.4)

  Por su parte, Panda fue desarrollado con el objetivo de producir nuevas técnicas de gestión de datos para aplicaciones que tuvieran un uso intensivo de E/S. Panda es una biblioteca de E/S diseñada para aplicaciones que ejecutaran en estaciones de trabajo ordinarias, en arquitecturas paralelas con memoria distribuida y en redes de estaciones de trabajo (NOWs). Panda está formado por un conjunto de clientes y servidores, uno por cada nodo de procesamiento y E/S respectivamente. La aplicación comunica con el cliente Panda a través de una interfaz de alto nivel. El cliente pasa a los servidores una descripción de alto nivel de la petición de E/S, que incluye la distribución en memoria y disco. Los servidores se encargan de atender la petición y devolver los resultados a los clientes. Esta biblioteca permite elegir una organización de datos eficiente entre los servidores y, por tanto, optimizar los accesos a los ficheros por parte de los nodos clientes.

  Jovian [BBS+94] es otra biblioteca de E/S para aplicaciones SPMD. Su principal optimización consiste en unir muchas peticiones de E/S pequeñas en una única petición grande. Para ello, distingue dos tipos de procesos: procesos de aplicación (AP: *Application Process*) y procesos de unión (CP: *Coalescing Process*). Todos los APs envían peticiones de E/S a CPs predeterminados y los CPs se encargan de llevar a cabo la E/S colectiva. A continuación, los CPs realizan las operaciones de E/S y transfieren directamente los datos a su correspondiente AP. Estas transferencias suponen una sobrecarga por la copia en memoria.

  Otras bibliotecas de E/S paralela son DRA (*Disk Resident Arrays*) [NF96], MIOS [TG96], VIPFS [HRC95], ChemIO [NFK98] o VIPIOS [BMS96].

  Las diferentes bibliotecas de E/S existentes proporcionan distintas APIs a las aplicaciones. Esta situación ha desembocado en una falta de estandarización de la E/S paralela. No existe una API estándar utilizada por todas las bibliotecas de E/S. El resultado es una falta de portabilidad de las bibliotecas de E/S paralela.

  Como se describe en la sección 2.5.1, MPI se ha convertido en el estándar "de facto" para la programación de arquitecturas multicomputador y multiprocesador, así como para los clusters de estaciones de trabajo.

Dentro de MPI, surge MPI-IO [CFF+95], [MPI96] que constituye la única iniciativa real de intento de estandarización de la interfaz paralela de E/S. MPI-IO comienza como un proyecto de investigación de IBM en 1994. MPI-IO se beneficia de dos características que MPI ya posee: la capacidad de definir conjuntos de procesos, a través de lo que en MPI se denominan comunicadores y la capacidad para especificar patrones de acceso complejos, a través de los denominados tipos de datos MPI (*MPI datatypes*). MPI-IO da soporte a múltiples operaciones de E/S paralelas y optimizaciones, tales como acceso a ficheros no contiguos, E/S colectiva, E/S asíncrona, preasignación de ficheros o punteros compartidos. En 1997, MPI-IO se incluye dentro del estándar MPI-2. MPI-IO debe implementarse sobre un determinado sistema de ficheros a fin de garantizar un buen rendimiento en el acceso en paralelo a los ficheros. No obstante, MPI-IO es sólo una especificación, por lo que es necesario utilizar una determinada implementación de la misma. ROMIO [TGL99b] es una implementación portable de MPI-IO, realizada en el *Argonne National Laboratory*. El componente que permite la portabilidad de esta implementación es ADIO (*Abstract Device Interface for Parallel I/O*) [RTL96], que ofrece una interfaz abstracta para E/S paralela. ROMIO es implementado encima de ADIO y sólo ADIO debe ser implementado de forma separada para los diferentes sistemas de ficheros.

- **Sistemas de ficheros paralelos.** Según [Sto98], un sistema de ficheros paralelo es aquél que elimina el cuello de botella de E/S agregando de forma lógica múltiples dispositivos de almacenamiento independientes y nodos de E/S mediante un sistema de almacenamiento de alto rendimiento. El ancho de banda puede incrementarse mediante **direccionamiento del disco independiente** (el sistema de ficheros puede acceder a datos de diferentes ficheros de forma concurrente) y ***declustering* de datos** (un solo fichero se puede acceder en paralelo).

  Un sistema de ficheros paralelo opera, al igual que los sistemas de ficheros distribuidos, independientemente de la aplicaciones ofreciendo una mayor flexibilidad y generalidad que las bibliotecas.

  Los sistemas de ficheros paralelos pueden ser clasificados según diferentes taxonomías. Una posibilidad es clasificarlos como sistemas comerciales o sistemas orientados a la investigación, a saber:

  - Sistemas comerciales tales como PIOFS [CFP+95], para IBM SPs, PFS [EK93], [RP95] para las Paragon, CFS [Pie89], [Nit92] para Intel o SFS [LIN+93], [BGST93] para Thinking Machines CM-5.
  - Sistemas de ficheros de investigación: Algunos sistemas de ficheros proporcionan interfaces de E/S paralelas portables y no propietarias, como PIOUS [MS94], [MS95] y PETSc/Chameleon I/O [GGL93]. El primero consiste en una interfaz paralela basada en PVM [GBD+94b]. El segundo puede ejecutar encima de sistemas de ficheros propietarios, así como de las plataformas p4[2] [BL92], PICL[3] (*Portable Instrumented Communication Library*)[GHPW90] y PVM.

  Vesta [CJF93], [CFPS93], [DCP95] es un sistema de ficheros que ejecuta en el sistema IBM SP1. Este sistema permite definir vistas sobre los ficheros, usar operaciones colectivas y asíncronas. Vesta es implementada en una arquitectura cliente-servidor, ejecutándose los servidores en los nodos de E/S. Éstos llevan a cabo *prefetching* y escritura retardada.

[2] p4 es una biblioteca que contiene macros y subrutinas desarrolladas por ANL para programación de máquinas paralelas. Al igual que PVM y MPI, p4 ofrece un modelo de programación maestro-esclavo

[3] PICL es una biblioteca de subrutina que implementa una interfaz de paso de mensajes genérica en variedad de multiprocesadores. PICL está obsoleto y ha sido sustituido por MPICL

PPFS [JER+95] es un sistema de E/S paralelo formado por un conjunto de módulos que controlan las características de *caching*, *prefetching*, consistencia de datos, patrones de acceso y *layouts* de ficheros. La interfaz proporcionada a los usuarios es similar a la tradicional interfaz POSIX.

Galley [NK96] es un sistema de ficheros paralelo construido en *Dartmouth College*. Galley tiene una arquitectura cliente-servidor, donde los nodos de E/S contienen un gestor de cache y un gestor de discos. No obstante, su interfaz es tan compleja que es difícil para un programador de aplicaciones realizar llamadas a funciones Galley a fin de obtener un alto rendimiento.

ParFiSys [CPdM+96], [CPdM+97b], [CPdM+97a] consiste en un sistema de ficheros paralelo que proporciona un conjunto de servicios de E/S para aplicaciones científicas y que permite explotar el paralelismo en sistemas de E/S distribuidos.

PVFS (*Parallel Virtual File System*) [CIRT00] es un sistema de ficheros paralelo, que permite que aplicaciones secuenciales y paralelas almacenen y accedan a ficheros distribuidos a través de un conjunto de servidores de E/S. PVFS ha sido desarrollado para *clusters* Linux, proporcionando un gran ancho de banda en operaciones de lectura y escritura concurrentes realizadas desde múltiples procesos o threads a un fichero común.

MOPI (*MOSIX Scalable Parallel Input/Output*) [ABS02] utiliza la capacidad de migración de procesos que tiene MOSIX [BO98] para acceder en paralelo a segmentos de datos que se encuentran distribuidos entre diferentes nodos. Para utilizar este sistema de E/S de forma eficiente se ha implementado el sistema de ficheros MFS (*MOSIX File System*), que proporciona una visión unificada de los ficheros distribuidos en los nodos del cluster MOSIX. A diferencia de la mayoría de los sistemas de ficheros en red, que permiten llevar los datos desde el servidor al cliente a través de la red, los algoritmos utilizados en MOSIX migran el proceso al nodo en el cual la información reside, eliminando de ese modo la sobrecarga de comunicación entre el proceso y el servidor de ficheros (salvo por el coste de la migración de los procesos).

Expand [GCPS02], [GCC+02] consiste en un sistema de ficheros paralelo que utiliza servidores NFS en la parte servidora, siendo el cliente el que proporciona el paralelismo. La arquitectura de E/S objeto de esta tesis tiene en común con este sistema de ficheros la arquitectura básica.

Otros sistemas de ficheros paralelos ampliamente referenciados en la bibliografía son Bridge [DSE88], nCUBE [DdR92], HFS [Kri94], Scotch [Gib95] y ParFiSys [CPdM+96], [CPdM+97b], [CPdM+97a].

El principal problema de las bibliotecas paralelas de E/S es que están concebidas fundamentalmente para el desarrollo de aplicaciones paralelas y no ofrecen una solución genérica para su uso en sistemas distribuidos. En cuanto a los sistemas de ficheros paralelos, su principal problema es que están especialmente pensados para máquinas paralelas y no se integran adecuadamente en entornos distribuidos de propósito general. Además, cada sistema de ficheros paralelo utiliza una estructura de fichero paralelo diferente, incompatible con la de otros sistemas.

4.3.2. Sistemas de almacenamiento de altas prestaciones (redes de almacenamiento)

El esquema tradicional para gestionar el espacio de almacenamiento en un sistema distribuido ha consistido en disponer de uno o varios servidores de disco accesibles a través de una red de área local, con los problemas que anteriormente se han descrito. Recientemente se ha propuesto la introducción de una nueva red dedicada exclusivamente al subsistema de almacenamiento: la SAN (Storage Area Network) [GNA+97]. Una SAN (véase la figura 4.2) enlaza los dispositivos de almacenamiento (como

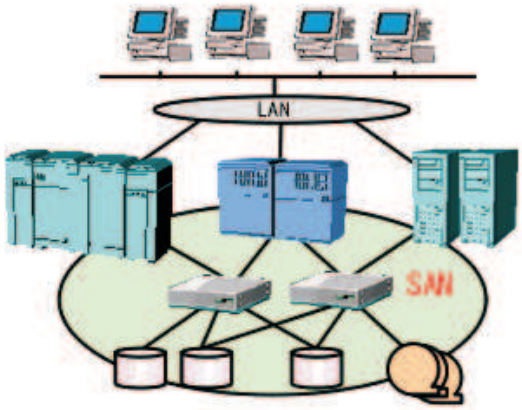


Figura 4.2: Configuración de un sistema con Storage Area Networks

por ejemplo discos, RAID y cintas) y los servidores con el objetivo de crear un gran espacio de almacenamiento que puede ser gestionado conjuntamente.

Con la introducción de la SAN distinguimos entre dos redes diferentes. Por un lado, la que une los clientes y los servidores puede ser una red de área local (e.g., Fast-Ethernet). De otro lado, la SAN se encarga de conectar los dispositivos de almacenamiento a los diferentes servidores que proporcionan la información a los clientes.

El problema de la SAN es su alto coste, lo que puede limitar su uso. No obstante, es una solución cada vez más utilizada.

4.3.3. Distribución de los datos en sistemas de ficheros paralelos

Uno de los aspectos clave que hay que resolver en un sistema de ficheros paralelo es cómo se lleva a cabo la distribución de los datos a través de los diferentes nodos. Existen varias formas de distribuir los datos de un fichero. Una posible clasificación de la distribución de los datos es descrita por García en [Gar98], a saber:

- **Distribución horizontal:** Se distribuye el fichero por bloques de datos. Se pueden distinguir dos tipos de distribución horizontal:

  1. **Agrupada:** Los bloques de un fichero se distribuyen de forma secuencial en uno o varios dispositivos. Con este tipo de distribución, se pueden tener tres tipos de ficheros:

     Ficheros autocontenidos: Ficheros cuyos bloques se almacenan en un solo disco.

     Ficheros extendidos: Ficheros que se extienden a más de un disco, si su tamaño sobrepasa la capacidad de uno o más dispositivos.

     Multificheros: Ficheros formados por subficheros, a los que se accede de manera independiente. Un multifichero es creado por un programa paralelo a partir de un número determinado de subficheros, que normalmente es igual al número de procesos del programa; cada subfichero se direcciona con independencia del resto y puede modificar su

tamaño sin afectar al resto. Los multificheros se benefician de las ventajas que ofrece un único fichero (un único nombre para todo el conjunto de datos) y las ventajas que ofrecen múltiples ficheros (la capacidad de acceder a los subficheros de forma independiente).

2. **Desagrupada:** Los bloques de un fichero se distribuyen siguiendo un determinado patrón de entrelazado entre los distintos dispositivos. Se pueden distinguir a su vez cuatro tipos de distribución desagrupada:

   Irregular: Ficheros distribuidos de forma aleatoria a través de los dispositivos.

   Circular: Ficheros distribuidos según un patrón circular.

   Vecino más cercano: Cada bloque del fichero se asigna al dispositivo más cercano.

   Función de usuario: Para determinar el acceso a cada bloque, se utiliza una función definida por el usuario.

- **Distribución vertical:** En este tipo de distribución, se fragmentan los bloques de cada fichero, asignando dicha fragmentación a los diferentes dispositivos de los que consta el sistema.

También se pueden utilizar distribuciones de datos híbridas.

4.3.4. Optimizaciones de la E/S paralela

El salto existente entre la velocidad de acceso a los dispositivos de almacenamiento secundario y la velocidad de las unidades de procesamiento ha llevado a los diseñadores hardware a intentar mejorar las capacidades del sistema de E/S proporcionando acceso a múltiples discos, ofreciendo procesadores de E/S y buses de E/S con un alto ancho de banda [dRC94]. No obstante, el software también juega un papel importante en el intento de minimizar este salto. Para ello, los sistemas de E/S paralela pueden aplicar diferentes técnicas de optimización que intenten reducir las latencias de E/S. Actualmente se utilizan diversas técnicas, entre las que destacan las siguientes:

- **Cribado de datos** (*d*ata sieving) [TGL98], [TGL99a]. Para reducir el efecto de una alta latencia de E/S es crítico hacer tan pocas peticiones al sistema de ficheros como sea posible. El cribado de datos es una técnica que permite hacer pocas peticiones grandes y contiguas al sistema de ficheros incluso si las peticiones del usuario suponen pequeños accesos no contiguos. La idea básica del cribado de datos es realizar grandes peticiones de E/S y extraer, en memoria, los datos que realmente se necesitan. Se leen más datos de los que realmente se requieren, pero aún así se incrementa el rendimiento de E/S.

- Uso de **operaciones colectivas**, en las cuales todos los procesos que forman parte de una aplicación cooperan en cada petición de E/S. Las peticiones de E/S individuales se combinan a fin de obtener una única petición de E/S mayor. De este modo, se incrementa de forma significativa el ancho de banda de E/S efectivo. Respecto a esta opción, se pueden distinguir diferentes técnicas que permiten llevarla a cabo:

  - **Agrupación de peticiones de E/S**(*grouping I/O requests*) [Nit92]: Esta técnica permite minimizar el número de acceso a discos, ordenando las peticiones de acuerdo a la ubicación de los datos en disco.
  - **E/S en dos fases** (*two-phase I/O*) [dRBC93]. La idea que subyace en esta técnica es llevar a cabo la E/S en dos fases: una fase de E/S y una fase de comunicación. En la fase de E/S, se lleva a cabo la operación de E/S que mezcla las peticiones de todos los procesos. Si la

petición no es contigua, se utiliza cribado de datos. En la fase de comunicación, los procesos redistribuyen los datos para lograr la distribución deseada. En las operaciones de lectura, la primera fase es la fase de E/S y la segunda fase es la de comunicación. En las operaciones de escritura el orden es el inverso.

- **E/S directa a disco** (*disk-directed I/O*) [Kot94]: Esta técnica permite que las operaciones de E/S colectivas sean enviadas a los procesadores de E/S, que se encargan de examinar dichas peticiones, realizando una lista de bloques de disco a transferir y ordenando la misma. Finalmente, se utiliza doble *buffering* y mensajes especiales para realizar la transferencia. Esta técnica supone la optimización del acceso a disco y un menor uso de memoria y CPU.
- **E/S directa a servidor** (*server-directed collective I/O*) [SCJ+95]: La filosofía básica de esta técnica consiste en que los servidores de E/S se encargan de gestionar de forma activa las operaciones de E/S. Esta técnica está basada en la técnica de la E/S directa a disco y consiste en que los servidores de E/S dirigen el flujo de las peticiones de E/S. Los servidores de E/S conocen la distribución de los datos a través de los nodos de computación y de los nodos de E/S. Y, por tanto, aprovechan este conocimiento para optimizar las operaciones de E/S. La técnica de E/S directa a servidor es la versión a nivel lógico de la técnica de E/S directa a disco.

En [Kan01] se describe una técnica basada en la E/S colectiva, denominada MCIO (*Multi-Collective I/O*), que permite optimizar los accesos de múltiples procesos a múltiples ficheros, teniendo en cuenta los patrones de acceso entre ficheros. La diferencia con la E/S colectiva es que ésta sólo permite combinar los accesos de diferentes procesos a un único fichero.

- Incremento de la funcionalidad de las interfaces tradicionales de operaciones de E/S de bajo nivel (`read()`, `write()`, ...) añadiendo nuevas **interfaces de alto nivel** (por ejemplo, una operación para realizar la lectura de una matriz).
- Modelización de patrones de acceso frecuentes, de forma que se optimice el acceso al disco en función del dominio de las aplicaciones.
- ***Caching* y *prefetching***: Estas dos técnicas permiten la reutilización de los datos de los ficheros, aprovechando la localidad de referencia que pueda darse entre procesos.
- Uso de ***hints*** que recojan información que permita optimizar los accesos de E/S.

A continuación, se describen en detalle estas tres últimas técnicas.

4.3.5. Patrones de acceso

El rendimiento de los accesos de E/S depende en gran medida de dos aspectos relacionados: la disposición de los datos (*data layout*) en los ficheros, que se denomina patrón de almacenamiento (*storage pattern*) y la distribución de los datos a través de los diferentes nodos, es decir, el patrón de acceso (*access pattern*). En aquellos casos donde ambos patrones no coincidan, permitir que cada nodo acceda de forma independiente puede llevar a que se realicen un gran número de peticiones de E/S pequeñas, lo que degrada la eficiencia de dichos accesos.

Por tanto, una buena disposición de los datos es crítica en la mejora del rendimiento de las operaciones de E/S. En sistemas tales como nCUBE [DdR92] o Vesta [CFPS93], el usuario tiene un mayor control sobre el *layout*.

Además, el conocimiento de los patrones de acceso permite utilizar dicha información de la forma más eficiente posible, incrementando el rendimiento global del sistema de E/S.

Se han llevado a cabo una gran cantidad de estudios sobre las cargas de los sistemas de ficheros, siendo éstos tradicionales, distribuidos o paralelos. En [Smi81] se lleva a cabo un estudio sobre el comportamiento de los *mainframes* de IBM respecto a los accesos a ficheros por parte de un editor interactivo a fin de predecir los efectos de la migración automática de ficheros. En [Flo86], [FE89] y [OCH+85] se estudian patrones de acceso a los ficheros para sistemas UNIX aislados. Respecto a sistemas distribuidos, en [BHK+91] se estudian patrones de acceso en el sistema distribuido Sprite [OCD+89]. [RBK92] analiza los patrones de acceso en un entorno comercial y sobre una plataforma VAX/VMS. Se han llevado a cabo varios estudios sobre cargas de trabajo de E/S científicas, tales como [dRC94], [MK91b] y [PP93]. También se han realizado estudios sobre cargas de trabajo de E/S de programas científicos paralelos [Cro89], [Kot93], [CK93], [CHKM93], [GGL93]. Dentro de este apartado, es destacable el trabajo realizado mediante el proyecto CHARISMA (**CHAR***acterizing* **I***/O in* **S***cientific* **M***ultiprocessor* **A***pplications*), que lleva a cabo un estudio de la carga de trabajo a nivel de sistema de ficheros de un entorno de producción paralelo [KN94], [PEK+94].

Las aplicaciones científicas utilizan ficheros grandes y tienen accesos que se caracterizan por su secuencialidad [MK91b], [GGL93], [PP93]. Muchos sistemas constan de sistemas de ficheros paralelos derivados del sistema operativo UNIX, lo que hace que tengan un rendimiento muy bajo, ya que UNIX fue diseñado para una carga de trabajo de propósito general [OCH+85], [Flo86], [FE89]. Los programas científicos paralelos utilizan patrones de acceso diferentes a las cargas de trabajo de un sistema monoprocesador o un sistema distribuido [NKP+95] [KN95]. Una de las soluciones que se han propuesto para resolver estas diferencias es el uso de la **E/S colectiva**. En un sistema con una interfaz tradicional al estilo de UNIX, las peticiones al sistema de ficheros se hacen de forma individual, incluso si la cantidad de datos requerida es pequeña. Por el contrario, un sistema que proporcione E/S colectiva permite la petición conjunta de muchos procesos.

En todo caso, estudios experimentales [CACR95], [RMA+96], [SR97], [SACR01] han demostrado que las aplicaciones paralelas tienen patrones de accesos bastante complejos, variables tanto espacial como temporalmente.

Por otro lado, han surgido algunos sistemas de ficheros enfocados a datos estrechamente relacionados con un campo específico. En este sentido, el sistema ChemIO [NFK98], utiliza como conjunto de datos de entrada datos relativos a aplicaciones químicas.

Panda [Sea96], [Cho99] prefiere ocultar los detalles físicos a las aplicaciones, definiendo lo que denomina esquemas Panda (*Panda schemas*) de forma transparente a las mismas, aunque también permite que la aplicación tenga un control explícito del sistema.

PPFS define un conjunto de patrones de accesos que deben ser especificados en el momento de abrir o crear un fichero paralelo [EKHM93].

Madhyastha y Reed [MR97] utilizan modelos de Markov ocultos (HMM: *Hidden Markov Models*) y redes neuronales para clasificar los patrones de acceso dentro de un fichero.

En [Kan01] se propone un compilador que permite mejorar el rendimiento de E/S de código científico. La responsabilidad del compilador consiste en analizar los patrones de acceso de los datos de aplicaciones individuales y determinar patrones de almacenamiento y estrategias de E/S adecuados para los primeros.

MPI-IO utiliza los denominados tipos de datos MPI (*MPI datatypes*) para describir la disposición de los datos tanto en memoria como en el fichero, de forma que se puede representar de forma compacta cualquier patrón de acceso no contiguo [TGL98], [TGL02]. El *layout* de los datos en memoria se especifica en cada función de lectura/escritura en MPI-IO. El *layout* de los datos en el fichero es definido por la vista del mismo. Cuando se abre un fichero, se define dicha vista. Un proceso puede modificar la vista de un fichero haciendo uso de la función `MPI_File_set_view()`.

4.3.6. *Caching* y *prefetching* de E/S

El empleo de una cache o almacenamiento intermedio en un sistema de E/S permite beneficiarse de la localidad temporal y espacial de los datos y de un acceso más rápido a los mismos, lo que permite aliviar el problema de la crisis de la E/S [Smi85].

Una cache almacena una copia de los datos recientemente utilizados en un dispositivo de almacenamiento más rápido que aquél en el que se encuentran los mismos. El uso de cache en el sistema mejora su rendimiento ya que reduce la contención en los accesos a los dispositivos, la carga en los nodos de E/S y la contención en la red de interconexión, al poder acceder a una gran parte de los datos de forma local.

Respecto al tipo de datos que almacena la cache en el sistema, podemos distinguir claramente dos tipos: datos correspondientes a bloques del fichero, hablando en este caso de **cache de bloques** y metadatos correspondientes a la información que necesita utilizar el sistema de ficheros para la gestión de los mismos, denominándose a dicha cache **cache de metainformación**.

La cache de bloques mejora el rendimiento de tres formas diferentes:

1. La cache de bloques utiliza el principio de proximidad de referencias en los accesos a un fichero, tanto proximidad temporal como espacial. En un sistema de ficheros paralelo la proximidad temporal sobre un fichero es una situación poco frecuente, debido a que predomina el acceso de tipo secuencial. Sin embargo, se da una alta proximidad espacial, debido al gran número de peticiones de E/S de tamaños pequeños [KN95], [PEK+94].

2. La cache de bloques permite la lectura adelantada (***prefetching***) de bloques de datos antes de que sean solicitados por las aplicaciones. Mejora el rendimiento de las operaciones de lectura, sobre todo en el caso de uso de patrones de tipo secuencial.

3. La cache de bloques permite el uso de políticas de **escritura diferida**, que retrasa la escritura de los datos modificados a los dispositivos de almacenamiento secundario. De este modo, se mejora el rendimiento de las operaciones de escritura.

Existen distintos parámetros de diseño en relación a la elección de la cache de bloques en sistemas de ficheros paralelos y distribuidos [Gar98]:

- **Localización de la cache:** Se puede situar la cache únicamente en los NES, pero eso supone el uso de un esquema centralizado y, por tanto, no escalable para un sistema de E/S distribuido-paralelo, ya que no reduce el acceso a recursos compartidos, tales como la red, la cache de los nodos de E/S y los dispositivos. No obstante, resuelve el problema de la coherencia de cache.

  La localización de la cache en los NP supone una solución más escalable, ya que reduce la contención en la red de interconexión, en los NES y en los dispositivos, lo que permite ofrecer un mayor paralelismo en el acceso a los datos. Por el contrario, complica la resolución del problema de coherencia de la cache.

- **Granularidad de la cache:** Se refiere al tamaño de los datos almacenados en la misma. El aumento de tamaño incrementa la probabilidad de aciertos en accesos posteriores y decrementa la utilización de la red de interconexión, aunque aumenta la latencia de las operaciones de E/S. No es adecuado para aplicaciones paralelas que accedan a datos no contiguos en el fichero.

- **Tamaño de la cache:** Viene determinado por los patrones de E/S utilizados. Incide en la tasa de aciertos de dicha cache. Para patrones secuenciales, basta una cache de tamaño pequeño, ya que la reutilización de bloques es muy pequeña. Para otros patrones, puede ser necesaria

la utilización de una cache de mayor tamaño. Es necesario conocer el comportamiento de las diferentes aplicaciones que utilicen el sistema de ficheros para calcular el tamaño óptimo de la cache, intentando optimizar la relación coste-rendimiento.

- **Política de reemplazo:** Es la política utilizada para decidir qué bloques se eliminan de la cache. Hay diferentes políticas, entre las que destacan FIFO (*First In First Out*), aleatoria, LRU (*Least Recently Used*) y MRU (*Most Recently Used*). Los factores que determinan la elección de la política de reemplazo incluyen los patrones de acceso a ficheros, las características físicas de los discos utilizados, la arquitectura multiprocesador o el uso de prioridades.

- **Políticas de actualización:** Determina cómo y cuándo volcar los bloques modificados al disco. Hay diferentes políticas de actualización:

  Escritura inmediata (*write-through*): Los bloques son escritos a disco en cuanto alguna aplicación los modifica.

  Escritura diferida (*write-back*): Los bloques se mantienen en la cache hasta que se requieren bloques libres en demanda a nuevas peticiones. Variantes de esta política son: *write-on-close*, que vuelca todos los bloques modificados a disco cuando se cierra el fichero y *write-full*, que retrasa la escritura a disco hasta que el buffer de la cache se llena.

  La escritura diferida reduce la latencia de las escrituras y permite eliminar las escrituras a disco de ficheros temporales de corta duración. Además, también mejora el rendimiento al permitir agrupar operaciones. No obstante, introduce un problema de fiabilidad en el sistema de ficheros.

- **Lectura adelantada:** También llamado *prefetching*. Consiste en la lectura de bloques por adelantado antes de que sean solicitados por las aplicaciones. De este modo, se puede lograr solapar el tiempo de E/S con el tiempo de cómputo de las aplicaciones.

Por otro lado, también se puede utilizar una estructura cache para la metainformación. Se denomina metainformación en un sistema de ficheros a la información que describe la estructura del sistema de ficheros de forma global y la estructura de cada fichero en particular, es decir, su nombre, atributos e información de direccionamiento del mismo.

La metainformación puede ser persistente, si se localiza en disco (ej: bloques de datos), o transitoria, si existe únicamente cuando se accede al fichero (ej: puntero de posición).

Algunos metadatos son visibles a las aplicaciones (ej: longitud de un fichero) y otros son ocultos (ej: distribución del disco).

Esta información puede almacenarse en caches. Las principales caches de metainformación que proporciona un sistema de ficheros son:

- **Cache de particiones**, que permiten un rápido acceso a la información que describe cada partición. Ejemplo: superbloque de UNIX.

- **Cache de atributos de ficheros**, que almacena los atributos de aquellos ficheros que se están utilizando. Ejemplo: i-nodo de UNIX y tabla de i-nodos.

- **Cache de nombres**, que optimiza la decodificación de nombres en un sistema de ficheros.

- **Cache de información de direccionamiento**, que almacena información que permite acceder a los bloques que forman el fichero. Ej: i-nodo en UNIX, que se complementa con el uso de bloques indirectos.

Las técnicas de *caching* y *prefetching* de ficheros se han convertido en técnicas estándar en el campo de los sistemas de ficheros secuenciales y distribuidos [GA94], [NWO88], [HKM+88]. En este tipo de sistemas, la técnica de *prefetching* más comúnmente utilizada consiste en leer secuencialmente el fichero de forma adelantada, lo que implica detectar si una aplicación accede de forma secuencial sobre un fichero y en caso afirmativo, leer los bloques de datos en dicho orden secuencial [MJLF84], [MK91a].

Un gran número de sistemas de ficheros distribuidos proporcionan dos niveles de cache, tales como Sun NFS, Sprite, AFS o CODA.

Por otro lado, las técnicas de *caching* y *prefetching* también se han implantado en los sistemas de ficheros paralelos a fin de optimizar el rendimiento de las operaciones de E/S [EK89], [KCF+96], [PGG+01].

Existen algunos trabajos que utilizan compresión de datos para el diseño de estrategias de *prefetching* óptimas [CKV93], [KV94], [VK96]; otro trabajo define estrategias de *prefetching* a través de un modelo probabilístico de las secuencias de peticiones [PSV94].

En [KCF+96] se describe un modelo teórico que permite caracterizar un sistema, a fin de aplicar las técnicas de *caching* y *prefetching*.

También se ha estudiado la combinación de las técnicas de *prefetching* y *caching* [CFKL95].

4.3.7. *Hints*

Los *hints* ("pistas") son una técnica ampliamente utilizada para mejorar el rendimiento del sistema, no restrigiéndose al campo de los sistemas de ficheros. Ya en 1983, Lampson describió su uso en sistemas operativos, tales como Alto o Pilot, en redes como Arpanet o Ethernet y en lenguajes como Smalltalk [Lam83].

En el contexto de los sistemas de ficheros, los *hints* se suelen utilizar como información histórica a fin de optimizar las técnicas de *caching* y *prefetching*. De hecho, es su uso más habitual. La política de reemplazo LRU, por ejemplo, se basa en el histórico de los accesos para elegir los datos a reemplazar. Por otro lado, se pueden utilizar *hints* en la fase de *prefetching* para decidir cuántos bloques se deben leer por adelantado. Como un ejemplo de esta práctica, en el sistema OSF/1 se realiza un *prefetching* de hasta 64 bloques cuando se detectan largas ejecuciones secuenciales [PGG+01]. En otros trabajos se detectan patrones de acceso más complejos para realizar un *prefetching* no secuencial dentro del fichero [KE91].

Existe un gran número de trabajos que se centran en inferir accesos futuros basados en accesos pasados [Kor90], [TD91], [GA94]. En [KE91], se utiliza el histórico de acceso para predecir la carga de E/S y por tanto, mejorar la fase de *prefetching*. Para ello Kotz y Ellis imprementaron predictores de patrones, que permiten predecir la carga de E/S futura.

Una clase alternativa de *hints* son aquéllos que permiten expresar conocimiento avanzado sobre un componente del sistema que pueda tener algún tipo de repercusión en otros, como puede ser el sistema de memoria virtual o la cache, de forma que el sistema recomienda determinadas políticas a fin de incrementar el rendimiento de las aplicaciones [Tri79], [CFL94].

Se pueden utilizar *hints* para expresar conocimiento más detallado. Por ejemplo, en el campo de las bases de datos, se puede utilizar esta información para la gestión de *buffers*, de modo que se utilicen los accesos de una consulta para determinar el número de *buffers* a asignar [CY89], [NFS91], [CR93].

En [PGS93] se distinguen dos tipos diferentes de *hints*, *disclosing hints* ("*hints* que muestran") y *advising hints* ("*hints* que aconsejan"). Los primeros describen el conocimiento necesario sobre el comportamiento de la aplicación, mientras que los segundos recomiendan cómo gestionar los recursos a fin de incrementar el rendimiento de las aplicaciones. Los *disclosing hints* tienen tres ventajas sobre

los *advising hints*:

1. La información que proporcionan es independiente de la implementación del sistema, por lo que se trata de un mecanismo de optimización de E/S portable.

2. Proporcionan información que permite tomar una decisión sobre la política a elegir, en lugar de especificar la política explícitamente. Esto supone una mayor robustez, ya que en caso de que no haya recursos suficientes para llevar a cabo la mejor política, se pueden escoger alternativas que permitan resolver el problema de forma parcial.

3. Este tipo de *hints* se expresa en términos de la interfaz del sistema.

En MPI-IO también se pueden especificar *hints* a través del uso de su interfaz [TGL02]. La estructura `MPI_Info` permite especificar pares de la forma `(clave, valor)`, que proporcionen información adicional sobre las operaciones de E/S. Existe un conjunto de claves reservadas, entre las que se encuentran *hints* relacionados con patrones de acceso o la disposición de los datos sobre dispositivos de E/S. Un ejemplo de estos *hints* lo constituye la clave `striping_unit`, que especifica la unidad de distribución de los ficheros a través de los diferentes dispositivos de E/S.

En definitiva, los *hints* proporcionan a los sistemas de E/S un papel más activo, ya que tradicionalmente éstos se han comportado de una forma pasiva, respondiendo exclusivamente a las peticiones de lectura y escritura de las aplicaciones. A partir de la información proporcionada por los *hints*, los sistemas de E/S pueden, por ejemplo, reconfigurarse a fin de incrementar el rendimiento de las aplicaciones.

Parte II

ESTUDIO Y RESOLUCIÓN DEL PROBLEMA

Capítulo 5

Diseño de MAPFS

5.1. Introducción

Los sistemas de ficheros proporcionan a las aplicaciones la capacidad de almacenar información de una forma persistente. Consisten en una capa software que proporciona una visión lógica de los dispositivos de almacenamiento, ofreciendo una interfaz de acceso adecuada a los mismos y los mecanismos apropiados de protección.

Inicialmente se utilizaron sistemas de ficheros tradicionales, los cuales sólo permiten acceso a los ficheros a los usuarios de dichos sistemas.

Con la proliferación de las redes de interconexión, los sistemas de ficheros distribuidos aparecieron como respuesta a la necesidad de compartir información distribuida entre diferentes servidores. En este tipo de sistemas, cada fichero se almacena en un servidor. No obstante, el acceso a los datos no se lleva a cabo en paralelo.

Ninguno de estos tipos de sistemas de ficheros resuelven el problema ya descrito de la *crisis de la E/S*. En esta situación, surge la E/S paralela, como un intento de resolver este problema. La E/S paralela y más concretamente los sistemas de ficheros paralelos permiten un acceso paralelo a los datos, proporcionando mejor rendimiento a las operaciones de lectura y escritura.

Como vimos en el capítulo 1.3, estos sistemas de ficheros adolecen de algunos defectos. El objetivo de esta tesis incluye el diseño de MAPFS, que consiste en un nuevo sistema de ficheros paralelo y multiagente, orientado a su uso en *clusters* de estaciones de trabajo y que intenta resolver los problemas anteriormente reseñados [PGC01]. MAPFS consiste en un módulo cliente que permite acceder en paralelo a los datos distribuidos entre varios servidores de ficheros. En una primera fase, MAPFS se ha implementado de forma que pueda emplear particiones distribuidas a través de varios servidores NFS.

Por otra parte, MAPFS se basa en una arquitectura multiagente que sirve como soporte para la adquisición de los datos y que le proporciona las características de tolerancia a fallos y *caching*, como servicios adicionales, que pueden ser ampliables por parte del sistema.

Además, MAPFS posee una interfaz que incluye operaciones tradicionales, colectivas, avanzadas, de tolerancia a fallos y de *caching*.

El objetivo del presente capítulo es la descripción del diseño de MAPFS tanto desde un punto de vista arquitectónico como desde el punto de vista del diseño de la entidad básica, el fichero. En la primera sección se analizan los requisitos de diseño del sistema de ficheros de una forma global. En la segunda sección se describe la arquitectura de MAPFS, haciendo énfasis en el aspecto modular del mismo. Finalmente, se describen de una forma detallada los dos subsistemas que componen el sistema MAPFS, el subsistema de ficheros y el subsistema multiagente.

5.2. Requisitos de diseño

Esta sección analiza los requisitos de diseño de MAPFS, concluyendo la necesidad de implementar dos subsistemas que se complementan, con el objetivo de proporcionar la funcionalidad requerida.

Los principales objetivos que sirven como guía para el diseño del sistema de ficheros MAPFS son:

1. Proporcionar una interfaz que permita utilizar el sistema como un cliente capaz de interactuar con un servidor de ficheros tradicional o distribuido, tal como NFS.

2. Realizar las peticiones de MAPFS a los servidores de ficheros en paralelo. De este modo, el sistema podrá beneficiarse de la característica de paralelismo.

3. Permitir la utilización de servidores heterogéneos, con distintos sistemas operativos y distintos sistemas de ficheros.

4. Incrementar la portabilidad del sistema de ficheros, reduciendo las partes dependientes de la implementación.

5. Incrementar la autonomía del sistema, ofreciendo diferentes tareas independientes y configurables de mantenimiento, tolerancia a fallos e incremento del rendimiento.

6. Realizar un diseño del sistema modular y flexible. De esta forma, será sencillo ampliar la funcionalidad del mismo y modificar sus características propias.

7. Utilizar clusters de estaciones de trabajo como plataforma subyacente, debido a que el escenario de computación actual ha cambiado en los últimos años, comenzando a utilizarse de forma masiva clusters de estaciones de trabajo, en lugar de los tradicionales supercomputadores especializados. MAPFS debe adaptarse también a un sistema de almacenamiento basado en SAN.

8. Definir un formalismo que permita gestionar y configurar de una forma dinámica los servidores que forman parte de la arquitectura. Este formalismo se denomina grupo de almacenamiento y se describe en el capítulo 7.

9. Proporcionar diferentes optimizaciones de E/S, tales como *caching*, *prefetching* o *hints*, ofreciendo flexibilidad a las aplicaciones para que puedan hacer uso de las mismas. Para ello, MAPFS ofrecerá una interfaz que permitirá adaptar dichas técnicas al dominio propio de las aplicaciones que lo utilicen. Estas optimizaciones se detallan en el capítulo 8.

Para cumplir estos objetivos, los requisitos de diseño que debe cumplir MAPFS, de forma que sirvan de guía para el diseño de la arquitectura y la modelización del fichero dentro del sistema integral, son principalmente:

- **Arquitectura cliente-servidor:** El paradigma utilizado es el modelo cliente-servidor. El trabajo se centrará en la creación de un cliente muy ligero que permita la paralelización, sin la modificación del servidor de ficheros correspondiente. De este modo, el sistema permite utilizar

servidores tradicionales o distribuidos, tales como NFS [SGK+85], [HKM+88], AFS [Kaz88], Calypso [BDK95] o cualquier otro servidor.

La arquitectura software cliente-servidor es una infraestructura modular, versátil y basada en paso de mensajes, que proporciona flexibilidad, interoperabilidad y escalabilidad a las soluciones software frente a un entorno centralizado. Estas características son deseables para nuestro sistema de ficheros MAPFS, por lo que utilizar un modelo cliente-servidor parece una decisión acertada.

No obstante, han surgido nuevas arquitecturas que ofrecen ventajas a la solución más simple de arquitectura cliente-servidor. Un tipo especial de arquitectura cliente-servidor lo constituye la arquitectura de tres niveles, en la cual se interpone una capa entre el cliente y el servidor, dividiendo de este modo las diferentes tareas del proceso global entre todas las capas o niveles que constituyen el modelo.

El diseño de tres niveles tiene ventajas sobre la arquitectura de dos niveles. En concreto:

- Debido a que se incrementa la modularidad, es más sencillo modificar o reemplazar uno de los niveles sin afectar al resto.
- Separar la lógica de la aplicación de la gestión de los datos permite un mejor equilibrio de carga.

La arquitectura de tres niveles es la base para la computación basada en componentes. El nivel o capa de interfaz de aplicación (API) sólo es una parte de la computación de N niveles. En la capa intermedia hay un conjunto de componentes que dan servicio tanto a otros componentes como al cliente final, y que interactúan entre sí. En este contexto se van a ubicar el servicio de tolerancia a fallos y el servicio de *caching*. De hecho, estos servicios se van a implementar a través del uso de **agentes** [JW98], [FG96].

Si nos fijamos en la arquitectura genérica de un sistema de E/S distribuido-paralelo, podemos distinguir los niveles funcionales, que se describen en el capítulo 4:

- nodo de proceso: NP;
- nodo de E/S: NES;
- controlador;
- dispositivo.

En el caso de MAPFS, los NP y NES no tienen una entidad tan diferenciada, ya que funden en cierto modo sus capacidades, al hacer uso de entidades intermedias que se van a encargar de realizar la tarea de paralelización. Ésta es una decisión de diseño condicionada por el hecho del uso de agentes en la tarea de recuperación de la información.

La ventaja de utilizar los clientes de este modo es que permite el uso de diferentes servidores (incluso aquéllos que no tienen ninguna característica de paralelismo), proporcionando de esta forma la característica de heterogeneidad, que permite que el cliente pueda interactuar con distintos servidores de ficheros.

- **Metainformación como estructura independiente del fichero:** La metainformación correspondiente al sistema de ficheros será independiente de la estructura del fichero, de forma que su gestión y tratamiento sean más modulares.
- **Clientes concurrentes:** MAPFS lleva a cabo la E/S paralela de forma inversa a la convencional. Generalmente los sistemas de ficheros paralelos utilizan servidores *multithread*, que se encargan

de atender las peticiones de E/S de forma concurrente. En el caso de MAPFS, son los clientes los que se encargan de realizar las diferentes peticiones a los servidores de forma concurrente. La eficiencia lograda dependerá del tiempo de servicio de los servidores, pero siempre es mayor que la de una petición secuencial, como queda demostrado en el capítulo 11.

- **Configurable:** Se debe permitir la configuración del sistema, de forma que sea parametrizable.

- **Tolerancia a fallos:** La distribución de los datos a través de un conjunto de discos y servidores hace al sistema más vulnerable. Por este motivo, el sistema de ficheros MAPFS debe ser capaz de proporcionar la capacidad de tolerancia a fallos a nivel de cliente. Cuando hablamos de tolerancia a fallos, nos referimos a **disponibilidad** de los datos [Jal94]. La disponibilidad es el porcentaje de tiempo que un sistema está proporcionando su servicio.

  En el caso de un entorno cliente-servidor, normalmente se proporciona la capacidad de tolerancia a fallos en el servidor. Este tipo de tolerancia es diferente dependiendo del tipo de servidor, a saber:

  Servidores con estado: En este caso, cuando se abre un fichero, el servidor almacena información y proporciona al cliente un identificador único a utilizar en las posteriores llamadas. Cuando se cierra un fichero, se libera la información. Las ventajas que ofrecen este tipo de servidores son:

  1. Mensajes de petición más cortos.
  2. Un mejor rendimiento (se mantiene la información en memoria).
  3. Facilitan la lectura adelantada.
  4. Capacidad por parte del servidor para analizar el patrón de accesos que realiza cada cliente.

  Servidores sin estado: Cada petición es autocontenida, es decir, contiene el identificador del fichero y posición. La ventaja fundamental de este tipo de servidor es que es más tolerante a fallos. Además, se reduce el número de mensajes, debido a que no son necesarias las operaciones de apertura y cierre de sesiones (normalmente llamadas `open` y `close`). Por otro lado, no se utiliza memoria en el servidor para almacenar el estado y, por tanto, es un servicio más escalable.

  MAPFS debe proporcionar la misma funcionalidad que un servidor sin estado, pero permitiendo que sea la parte cliente la que se encargue de la recuperación de los datos en caso de fallo en la parte del servidor.

  La idea de utilizar la metodología de agentes para dotar al sistema de tolerancia a fallos radica en la idea de que dicha característica se gestione de forma distribuida e independiente del cliente MAPFS.

- **Uso de distintos niveles de cache:** El sistema de ficheros MAPFS debe explotar la localidad temporal y espacial de los datos que requieren los clientes. Por ese motivo, debe proporcionar una estructura cache que contenga los datos más recientes.

  La tecnología de agentes también se puede utilizar a este nivel, de forma que se logre independencia en la gestión de las caches del sistema.

- **Autonomía:** La autonomía es una característica que permite a una entidad actuar por sí misma y en nombre de otras entidades. Se trata de una característica muy adecuada en sistemas distribuidos que incluyen diferentes funcionalidades adicionales y configurables. Por tanto, sería

deseable que el sistema tuviera procesos autónomos que se encargaran de tareas tales como adelantar peticiones sobre el sistema de ficheros o gestionar la tolerancia a fallos.

- **Modelo de fichero:** MAPFS debe utilizar un modelo de fichero que permita una distribución de datos sencilla y eficiente, pudiendo basarse en el modelo de ficheros de los sistemas de ficheros del servidor para construir su propio modelo. Como ejemplo, en el caso de utilizar servidores NFS sobre el sistema de ficheros UNIX, el modelo de fichero MAPFS debe poder construirse a partir del modelo del sistema de ficheros de UNIX, para de este modo, poder proyectarse de forma adecuada sobre el conjunto de servidores.

Según los requisitos establecidos, se pueden diferenciar dos aspectos de la arquitectura de E/S, objeto de esta tesis:

1. Aspectos relacionados con las características intrínsecas de un sistema de ficheros paralelo.
2. Aspectos relativos a la distribución de los datos y del procesamiento de los mismos. Debido a que este trabajo tiene como objetivo principal el uso de la agencia dentro del campo de la E/S, parece apropiado distinguir en el diseño de la arquitectura un módulo encargado de resolver la problemática asociada.

Estas dos líneas distinguen los dos subsistemas en los que se divide el sistema global, subsistema de ficheros y subsistema multiagente. Las secciones 5.2.1 y 5.2.2 muestran la idiosincrasia de cada uno de estos subsistemas.

5.2.1. Implicaciones para el diseño de sistemas de ficheros

Existen una serie de características que hace que un sistema de ficheros sea portable y ofrezca un gran rendimiento. Algunas de estas características que se presentan en [TGL99b] son:

- Acceso de gran rendimiento a ficheros. El sistema de ficheros debe diseñarse de forma que permita un acceso de gran rendimiento desde múltiples procesos a un fichero común. Esto implica que las peticiones concurrentes (y en particular las escrituras) no sean serializadas dentro del sistema de ficheros.

  Debido a que varios clientes pueden acceder de forma concurrente a la información, es necesario resolver los problemas de concurrencia generados por este proceso. Para ello es necesario tener un control de la sincronización. Debido a que todas las características de optimización y funcionalidad adicionales del sistema de ficheros (*caching*, tolerancia a fallos) son proporcionadas por el sistema multiagente asociado, también parece adecuado integrar el servicio de sincronización en el mismo sistema. Esto hace que los mecanismos utilizados sean independientes del sistema de ficheros e incluso que las políticas utilizadas dentro de esos mecanismos se puedan modificar de una forma más sencilla [1].

- Soporte a ficheros mayores de 4 Gbytes (2^{32}). Es importante que el sistema de ficheros sea capaz de soportar ficheros grandes. Esto implica que la interfaz del sistema de ficheros y las estructuras de datos internas deben utilizar enteros de 64 bits para representar los desplazamientos de los ficheros.

- Modelo de ficheros genérico. En el caso de MAPFS, vamos a basarnos en dos modelos diferentes de ficheros:

[1] Es una de las ventajas del diseño modular.

1. Modelo de subficheros: Este modelo corresponde según la taxonomía anterior a la distribución de datos horizontal de multificheros, descrito en la sección 4.3.3.
2. Modelo de distribución cíclica: La distribución del fichero lógico a través de los dispositivos físicos se realiza haciendo uso de una unidad de reparto para su "troceado" (*striped*). Este modelo corresponde a la distribución vertical, descrito también en 4.3.3.

En el presente trabajo se debe dar cabida a ambos modelos, de forma que el sistema resultante sea lo más genérico posible.

- Configuración de la distribución de datos de los ficheros. Será necesario establecer mecanismos que permitan controlar y modificar la distribución de los datos de los ficheros dependiendo de los patrones de acceso de las aplicaciones.
- Políticas de *caching/prefetching* variables. El sistema de ficheros debe detectar y adaptarse automáticamente a patrones de acceso cambiantes y/o proporcionar una interfaz para el usuario que especifique los patrones de acceso o las políticas de *caching/prefetching*. Se pueden utilizar *hints* para almacenar esta información adicional y de este modo mejorar dichas políticas [GCPdM99].
- Interfaz extendida frente a sistemas de ficheros tradicionales. La interfaz del sistema de ficheros paralelo MAPFS proporciona un conjunto de operaciones adicionales, utilizadas dentro del entorno de los sistemas de ficheros paralelos. Se pueden distinguir tres tipos:

  1. Operaciones para accesos no contiguos. Para acceder a datos no contiguos, se puede hacer una lectura contigua de una cantidad mayor de datos y una posterior criba. Esta operación se puede llevar a cabo de una forma eficiente. No obstante, es mejor llevar a cabo esta criba de datos dentro del sistema de ficheros (cuando ésta se hace dentro del sistema de ficheros, no existe ninguna diferencia respecto al *caching* regular; los datos leidos/escritos extra pueden quedar en la cache y no necesitan ser descartados). Por este motivo, el sistema de ficheros debe proporcionar una interfaz que soporte acceso no contiguo [RTL96], que permite incrementar el rendimiento de aplicaciones que utilizan diferentes patrones de acceso. Se desea además que el sistema se pueda monitorizar. Una de las métricas interesantes es aquélla que nos permite evaluar el espacio desperdiciado por la operación previamente comentada.
  2. Operaciones de lectura y escritura no bloqueantes. Al utilizar estas funciones, el proceso no se bloquea durante las operaciones de lectura y escritura. Estas rutinas no bloqueantes tienen que devolver un objeto o *handle* que se puede utilizar para comprobar la finalización de la operación.
  3. Operaciones de E/S colectiva. Mediante estas operaciones, los procesos que forman parte de una aplicación cooperan en cada petición de E/S. Las peticiones de E/S individuales se combinan en una única petición de E/S, incrementándose de este modo el ancho de banda de E/S efectivo de forma significativa.

Aparte de los requisitos propios del sistema de ficheros, de forma paralela se deben enumerar los requisitos del sistema multiagente encargado de la adquisición de los datos y otras funcionalidades.

5.2.2. Implicaciones para el diseño del sistema multiagente

En esta sección se desea proporcionar las líneas a seguir en el diseño del sistema multiagente, que se va a encargar de la recuperación de la información en el sistema de ficheros paralelo.

La naturaleza del problema de la creación de un sistema de ficheros paralelo puede ser resuelto de una forma más natural a través de un sistema distribuido, ya que una de las características en las que se basa este tipo de sistema de ficheros es la distribución de los datos. Un sistema multiagente es inherentemente distribuido, lo que facilita la tarea de integración entre dicho sistema y el sistema de ficheros paralelo. No obstante, los sistemas distribuidos se caracterizan por ofrecer soluciones más complejas, aunque mucho más adaptables al dominio del problema.

El sistema multiagente de MAPFS consiste en un conjunto de agentes, que poseen las siguientes características:

- El sistema multiagente debe gestionar los agentes de los que consta de una forma transparente.
- La característica de movilidad hace posible la transferencia de código entre los distintos nodos que forman el sistema. Esta transferencia puede ser muy útil en cualquier sistema de recuperación de información y de forma particular en la construcción de MAPFS. Los agentes móviles son útiles en tres áreas diferentes. Una de las áreas es la computación móvil, tal como los portátiles o las PDAs. El segundo campo es la obtención dinámica de software. El tercer campo es el de los sistemas de recuperación de información (*information retrieval*). MAPFS se podría beneficiar de esta última característica. De hecho, los agentes móviles presentan una serie de características que les hace muy adecuados en el uso de un sistema distribuido. Algunas de ellas quedan referenciadas en [LO98], a saber:
  - Reducción del tráfico por la red: El tráfico generado en los sistemas distribuidos suele ser un cuello de botella. Si se hace uso de agentes móviles, se puede evitar en gran medida parte del tráfico. La idea es migrar la computación hacia los datos, en lugar de los datos hacia el lugar de cómputo. Surge una panorámica completamente distinta. El transporte de los mensajes debe incluir el estado y el código.
  - Mejora de la latencia: Algunos sistemas de tiempo real no pueden ejecutarse de forma distribuida, debido al aumento de la latencia que ocasiona el uso de la red. Con el uso de los agentes móviles, esta limitación puede vencerse, debido a que los agentes móviles ejecutarán de forma local. Son los propios agentes los que migran a los diferentes nodos.
  - Ejecución asíncrona y autónoma: Los agentes móviles pueden actuar de forma asíncrona y autónoma al proceso que les creó. De hecho, pueden desconectarse y posteriormente volver a conectarse.
  - Se adaptan de forma dinámica: Los agentes móviles pueden distribuirse a través del sistema de forma que se adapten a las características cambiantes del mismo.
  - Son naturalmente heterogéneos: Los agentes móviles suelen ser independientes de la máquina y de la capa de transporte. Esta es una característica muy deseable, ya que los sistemas distribuidos suelen ser heterogéneos.
  - Son robustos y tolerantes a fallos: Los agentes móviles tienen la habilidad de moverse entre diferentes nodos, por lo que el hecho de que caiga un nodo en la red no implica que la aplicación también falle. Los agentes móviles pueden ser lanzados a diferentes nodos para continuar su ejecución. Esto hace que sea más sencillo programar aplicaciones distribuidas tolerantes a fallos y robustas haciendo uso de los agentes móviles.

  Todas estas características son deseables para la adquisición de datos en un entorno distribuido. No obstante, los agentes móviles tienen como principal desventaja su complejidad, lo que hace que sea difícil su integración en cualquier sistema. MAPFS implementa una solución intermedia, como se verá en el capítulo 9.

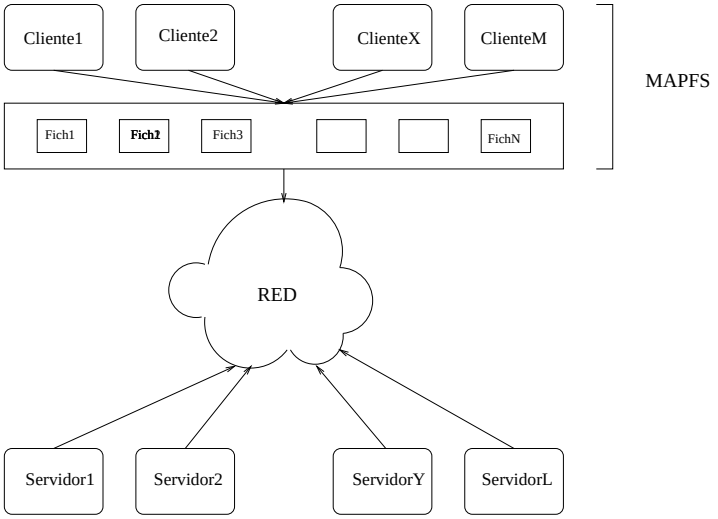


Figura 5.1: Comunicación entre MAPFS y servidores convencionales a través de la red

- Para resolver el problema de la adquisición de la información de una forma transparente y eficiente, se va a proceder a definir una jerarquía de agentes, distribuyendo las tareas entre los diferentes agentes.
- El uso de un sistema multiagente implica una coordinación entre los agentes que lo forman [LF97b], [Nec94]. Por tanto, será necesario establecer un protocolo de comunicación, así como un modelo de coordinación entre los agentes que se encargan de la recuperación de la información.

5.3. Arquitectura de MAPFS

Según los requisitos descritos anteriormente, la arquitectura del sistema de ficheros MAPFS es una arquitectura cliente-servidor que debe proporcionar la funcionalidad necesaria a nivel de cliente, pudiendo conectarse a cualquier servidor convencional que le proporcione la capacidad de almacenamiento. El paralelismo debe proporcionarse en el cliente.

El sistema MAPFS se encuentra ubicado en el cliente, en el espacio de usuario (desacoplado del kérnel). MAPFS modela de forma lógica un "servidor ligero" de ficheros en el cliente que, de forma transparente al usuario, sea capaz de conectarse a un servidor existente, como por ejemplo NFS, a través de la red de una forma transparente al usuario (véase figura 5.1).

Los ficheros se almacenan finalmente en varios servidores, que constituyen la parte servidora de la arquitectura subyacente. Para gestionar los servidores, éstos se agrupan de forma lógica. A estos grupos en el sistema de ficheros MAPFS se les denomina **grupos de almacenamiento** y se describen en el capítulo 7.

Si nos centramos en NFS como sistema de ficheros del servidor, MAPFS consiste en un sistema de ficheros paralelo que emplea particiones distribuidas a través de varios servidores NFS. La figura 5.2 muestra la arquitectura del sistema propuesto en este caso.

Como se ha descrito en la sección anterior, el sistema MAPFS se puede dividir de una forma natural en dos subsistemas: subsistema de ficheros, denominado MAPFS_FS y subsistema multiagente, denominado MAPFS_MAS. En la figura 5.3 queda representada la relación entre ambos subsistemas.

Ambos subsistemas se complementan a fin de lograr el objetivo de MAPFS, es decir, realizar la

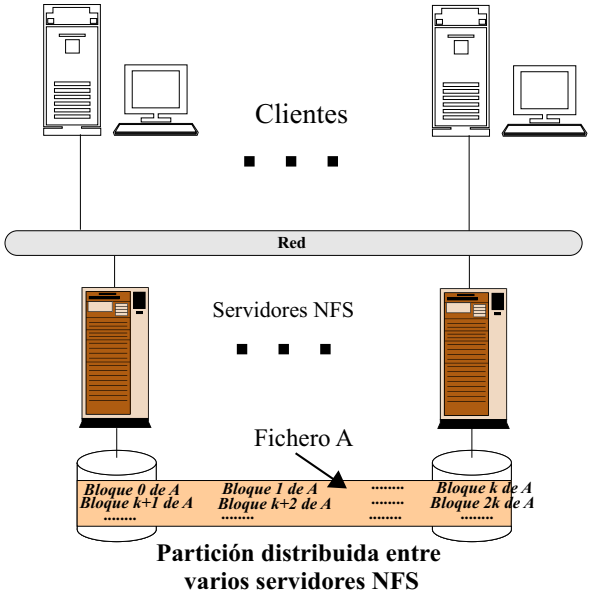


Figura 5.2: Arquitectura del sistema de ficheros paralelo propuesto mediante uso de NFS

adquisición de datos de una forma eficiente y transparente al usuario. En las secciones 5.4 y 5.5 se detallan ambos sistemas.

Por otro lado, MAPFS debe proporcionar una interfaz de aplicación, así como la capacidad de conexión a través de una red en un entorno distribuido. Esto se lleva a cabo mediante la **Interfaz de usuario** y el **Gestor de comunicaciones** respectivamente.

Además, el sistema de ficheros debe utilizar un esquema de redundancia que permita tolerancia a fallos al estilo de los sistemas RAID. Esta funcionalidad se lleva a cabo mediante el uso del **Gestor de Tolerancia a Fallos**. Este gestor se comunica con una determinada agencia. Una agencia no es más que un soporte *software* donde habitan varios agentes. Estos agentes permitirán proporcionar cierta autonomía al sistema para lograr dicha característica.

Finalmente, MAPFS se encarga de gestionar una cache propia del sistema de ficheros. El módulo encargado se denomina **Gestor de cache** y se comunica con una cache, así como con el sistema de agentes.

Para realizar el ensamblaje de todas las funciones del sistema de fichero, así como modelar un fichero, hacemos uso del **Gestor de ficheros**, que es el núcleo central del sistema de ficheros.

En la figura 5.4 aparece la estructura de un cliente MAPFS, formado por todos los módulos de los que se compone.

5.3.1. Interfaz de usuario

La interfaz de usuario es el módulo que proporciona acceso a la funcionalidad del sistema de ficheros MAPFS. En la sección 5.4 aparece descrita de forma detallada la interfaz del sistema. Arquitectónicamente, la interfaz de usuario es un módulo que se comunica con el gestor de ficheros, que es el que contiene el modelo del fichero utilizado por MAPFS. El gestor de ficheros es el que se encarga de realizar el conjunto de operaciones que la interfaz de usuario proporciona. El principal objetivo de dicha interfaz es aislar a los procesos que utilizan MAPFS de la implementación y del modelo interno del fichero.

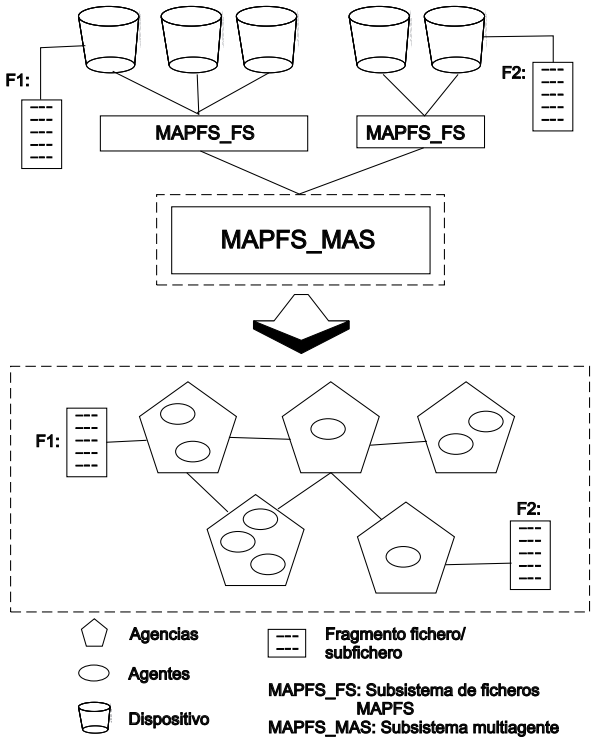


Figura 5.3: Subsistemas que forman el sistema de ficheros MAPFS

5.3.2. Gestor de comunicación

Este módulo permite gestionar la comunicación del sistema MAPFS con el servidor de ficheros. Este módulo tiene tres tareas principales:

- Realizar la transferencia de información y metainformación del sistema MAPFS.
- Encargarse de la comunicación entre el gestor de cache y el servidor de ficheros para la actualización de la cache.
- Realizar la distribución de agentes a través de la red formada por el sistema distribuido MAPFS.

Por este motivo, el gestor de comunicación queda enlazado al gestor de ficheros, al gestor de cache y al sistema multiagente.

5.3.3. Gestor de tolerancia a fallos

Este gestor permite proporcionar tolerancia a fallos al sistema de ficheros. El sistema de ficheros debe utilizar un esquema de redundancia que permita tolerancia a fallos al estilo de los sistemas RAID. El sistema de archivos debe permitir esta posibilidad cuando se cree un archivo, de forma que la característica de tolerancia se establezca a nivel de archivo.

Para gestionar la tolerancia, se utilizan agentes que forman parte del sistema multiagente asociado a MAPFS. A dichos agentes se les denomina **agentes de tolerancia**. Por este motivo, el gestor de tolerancia a fallos está conectado al sistema de agentes. También se debe comunicar con el gestor de ficheros debido al hecho de que la tolerancia se establece a nivel de archivo.

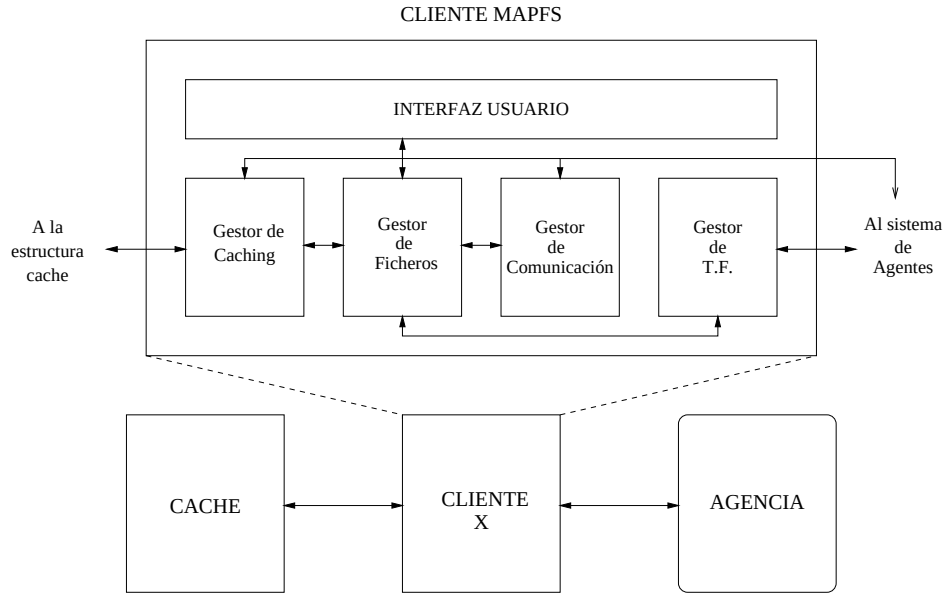


Figura 5.4: Módulos que constituyen un cliente MAPFS

5.3.4. Gestor de cache

El empleo de una cache o almacenamiento intermedio en el sistema de E/S permite beneficiarse de la proximidad de los datos y de un acceso más rápido a los mismos. Una cache tiene una copia de los datos más recientemente utilizados por el sistema en un dispositivo más rápido que el dispositivo de almacenamiento original de los mismos. Esta característica es aún más deseable en el caso de MAPFS, ya que al utilizar una cache, el sistema tiene menor dependencia del servicio que ofrezca el servidor convencional o distribuido contra el cual actúe el cliente MAPFS. No obstante, el uso de cache en la parte cliente crea un problema de coherencia importante, que es necesario resolver. De nuevo, es necesario utilizar un conjunto de agentes que forman parte de la agencia o sistema multiagente del sistema de ficheros, los cuales se encargan de la gestión de la cache del mismo. Los agentes que realizan esta tarea se denominan **agentes de cache** y son los responsables de utilizar un protocolo de coherencia de cache así como de controlar la transferencia de datos entre los dos dispositivos de almacenamiento, el disco del servidor y la memoria del cliente.

El gestor de cache se comunica con el gestor de ficheros, puesto que se encarga de optimizar el acceso a los mismos, con la estructura que sirve de cache al sistema y por tanto almacena la información correspondiente, con el sistema de agentes que se encargan de la gestión de la cache y con el gestor de comunicación, como vimos anteriormente.

5.3.5. Gestor de ficheros

Es el módulo encargado de modelizar la entidad básica del sistema de ficheros, es decir, el propio fichero. Por lo tanto, se comunica con el resto de los módulos para permitirles acceso a la funcionalidad primaria.

5.4. Subsistema de ficheros de MAPFS

El núcleo central del sistema de ficheros MAPFS lo constituye el subsistema de ficheros, también denominado MAPFS_FS. Este subsistema se encarga de implementar la interfaz del sistema de ficheros, proporcionando paralelismo y características adicionales que incrementan el rendimiento y la funcionalidad del mismo.

La funcionalidad que proporciona el sistema de ficheros MAPFS queda reflejada en la interfaz del mismo.

La mayoría de los sistemas de ficheros paralelos existentes utilizan una interfaz al estilo UNIX y se limitan a ofrecer operaciones de apertura, cierre, lectura, escritura y posicionamiento de ficheros, no mostrando a los usuarios la estructura paralela de los ficheros, ni permitiendo ofrecer optimizaciones del acceso y uso de los mismos, del cual se podrían beneficiar si no se ocultara dicha característica.

Estas interfaces tan simples no son apropiadas en sistemas paralelos, cuyas aplicaciones realizan una gran cantidad de accesos concurrentes a ficheros y tienen patrones de acceso más complejos.

Por otro lado, dada la característica de tolerancia a fallos con la cual se desea dotar a MAPFS, también será necesario ofrecer operaciones relativas a dicha característica.

Extendiendo la interfaz de los sistemas de ficheros tradicionales y distribuidos, MAPFS va a proporcionar cinco tipos de operaciones, que se enumeran a continuación. Todas las operaciones recogidas en esta sección son descritas de forma detallada en el apéndice A.

1. **Operaciones básicas:** Se trata de las operaciones típicas de un sistema de ficheros. En principio, MAPFS permite la utilización de un subconjunto de las primeras:

   - `mapOpen()`: operación de apertura de fichero.
   - `mapClose()`: operación de cierre de fichero.
   - `mapCreat()`: operación de creación de fichero.
   - `mapReadContig()`: operación de lectura contigua de ficheros.
   - `mapWriteContig()`: operación de escritura contigua de ficheros.
   - `mapSeek()`: operación de modificación de puntero.
   - `mapFcntl()`: operación para establecer u obtener información sobre un fichero abierto.

2. **Operaciones avanzadas:** En este apartado se contemplan todas aquellas operaciones adicionales que presentan los sistemas de ficheros paralelos y que le permiten añadir funcionalidad y mejora de rendimiento al mismo, excepto las operaciones de E/S colectiva, que se han distinguido en su propio apartado.

   Las aplicaciones paralelas necesitan frecuentemente leer o escribir datos que estén localizados de forma no contigua en los ficheros e incluso en memoria. Las operaciones `mapReadStrided()` y `mapWriteStrided()` son versiones bloqueantes de las llamadas `read()` y `write()` que soportan cualquier tipo de acceso no contiguo que pueda expresarse mediante los parámetros que reciben.

   La interfaz del sistema de ficheros MAPFS proporciona versiones no bloqueantes de todas sus llamadas de lectura y escritura.

   Los nombres de este tipo de operaciones se forman añadiendo el sufijo `I` al nombre de la operación original. Como ejemplo, la operación `mapIreadContig()` es la versión no bloqueante de la operación `mapReadContig()`.

   Para operaciones no bloqueantes, se establece un mecanismo de *callback*, de forma que se pueda especificar una operación que se lleve a cabo cuando una operación no bloqueante haya finalizado.

La interfaz del sistema de ficheros MAPFS proporciona versiones con posibilidad de establecer *callbacks* para todas las operaciones no bloqueantes de lectura y escritura.

Los nombres de este tipo de operaciones se forman añadiendo el sufijo `Cal` al nombre de la operación original. Como ejemplo, la operación `mapCalReadContig()` es la versión bloqueante de la operación `mapReadContig()`.

3. **Operaciones colectivas:** Se trata de operaciones orientadas a proporcionar paralelismo al sistema. Para ello hacemos uso del concepto de E/S colectiva, muchas veces utilizado en la bibliografía de sistemas de ficheros paralelos.

   Si procesos que realizan transferencias de grandes cantidades de datos, lógicamente relacionados, hacen uso de la interfaz que proporciona un sistema de ficheros tradicional, están obligados a hacerlo a través de múltiples peticiones pequeñas no contiguas, perdiéndose el sentido de una única transferencia grande. Por el contrario, si se hace uso de una interfaz colectiva, se permite a dichos procesos realizar una única petición grande, lo que permite que el sistema de E/S pueda incrementar el rendimiento de dicha transferencia de datos.

   Para permitir el uso de E/S colectiva, MAPFS proporciona versiones colectivas de todas las rutinas de lectura y escritura, tanto operaciones contiguas como no contiguas. Una rutina colectiva debe ser invocada por todos los procesos pertenecientes al grupo que abra el fichero. Sin embargo, una rutina colectiva no implica necesariamente una barrera de sincronización entre dichos procesos. Eso dependerá de la semántica de las operaciones.

   Los nombres de este tipo de operaciones se forman añadiendo el sufijo `Coll` al nombre de la operación original. Como ejemplo, la operación `mapReadContigColl()` es la versión colectiva de la operación `mapReadContig()`.

4. **Operaciones misceláneas:** Algunas operaciones adicionales para la gestión de ficheros son las siguientes:

   - `mapDelete()`: Operación para eliminar un fichero.
   - `mapResize()`: Operación para redimensionar un fichero.
   - `mapRename()`: Operación para modificar el nombre de un fichero.
   - `mapInit()`: Operación para abrir una sesión, inicializando parámetros de configuración y estructuras de datos.
   - `mapFinish()`: Operación para cerrar una sesión, liberando estructuras de datos.

5. **Operaciones de tolerancia a fallos y cache:** Las características de tolerancia a fallos y *caching* deben proporcionarse de una forma transparente. Por tanto, los usuarios no pueden utilizar estas operaciones y no aparecen en la interfaz de usuario de MAPFS. Son operaciones internas, que lleva a cabo el sistema multiagente. Estas operaciones consisten en:

   - Operaciones relacionadas con la recuperación de datos (disponibilidad de los datos).
   - Sincronización entre la cache y el dispositivo de almacenamiento secundario.
   - Coherencia de cache.

   No obstante, MAPFS ofrece operaciones a las aplicaciones de usuario para configurar *hints* que permiten incrementar el rendimiento de las operaciones de *caching* y *prefetching* (véase la sección 8.5).

La interfaz del sistema de ficheros MAPFS se ha diseñado en base a la interfaz ADIO [RTL96]. ADIO consiste en una interfaz de bajo nivel estándar para E/S paralela. Esta interfaz no está pensada para utilizarse directamente por programadores de aplicaciones, sino a nivel del sistema operativo, por desarrolladores de bibliotecas.

5.4.1. Modelo de fichero en MAPFS

El fichero es el objeto básico y fundamental de un sistema de ficheros. Por tanto, es necesario en todo sistema de ficheros diseñar la estructura de un fichero genérico, que debe contener la metainformación necesaria para describir el mismo. A esta estructura se le va a denominar *map-node*. La información que a priori va a ser necesaria incluye:

Nombre global del fichero: Este nombre debe proporcionar transparencia de localidad. El nombre global debe representar de forma unívoca el fichero al que referencia. Esta información al igual que ocurre en UNIX no debe encontrarse en el *map-node*, sino que debe encontrarse en las entrada del directorio que contenga dicho fichero. La diferencia estriba en el caso del modelo de subficheros, el cual podría tener diferentes nombres lógicos en cada uno de los nodos en los que se distribuye.

Tamaño del fichero: Representa el tamaño en bytes del fichero.

***F*lag para diferenciar el tipo de modelo:** *Flag* que diferencie el tipo de modelo que se va a utilizar en la paralelización del acceso a los ficheros (modelo de subficheros o modelo de distribución cíclica).

Posición: Apuntador a la posición del fichero en un dispositivo. Para determinar dicha posición, es necesario tener almacenado el dispositivo y posición relativa a ese dispositivo en el cual se encuentra el apuntador, en el caso de un modelo de distribución cíclica y el nombre del subfichero y posición relativa a dicho subfichero en el caso de un modelo de subfichero.

Protección: Control de acceso y permisos para la realización de las diferentes operaciones sobre el fichero.

Tiempos: Tiempos de creación, de acceso y de modificación, al estilo del sistema de ficheros de UNIX.

Identificación de usuario: Identificación del dueño y del grupo al que pertece el dueño del fichero.

***Hints* o pistas:** Se pueden establecer pistas para incrementar el rendimiento de acceso y modificación de los ficheros. Estas pistas pueden ser proporcionadas por la aplicación o bien pueden ser construidas por el sistema multiagente asociado.

Estructura que apunta a los datos del fichero: Esta estructura depende del tipo de modelo anterior y, por tanto, del valor que toma el *flag*. Si se trata de un modelo de distribución cíclica, esta estructura consiste en una lista con la localización del fichero a través de los múltiples dispositivos (en concreto, una lista encadenada de posiciones a través de los distintos dispositivos que forman el sistema). Si se trata de un modelo de subficheros, esta estructura consiste en una lista de los subficheros y tamaño de las porciones correspondientes a dichos subficheros por los que está formado el fichero original.

En la figura 5.5 se puede ver representado el *map-node*.

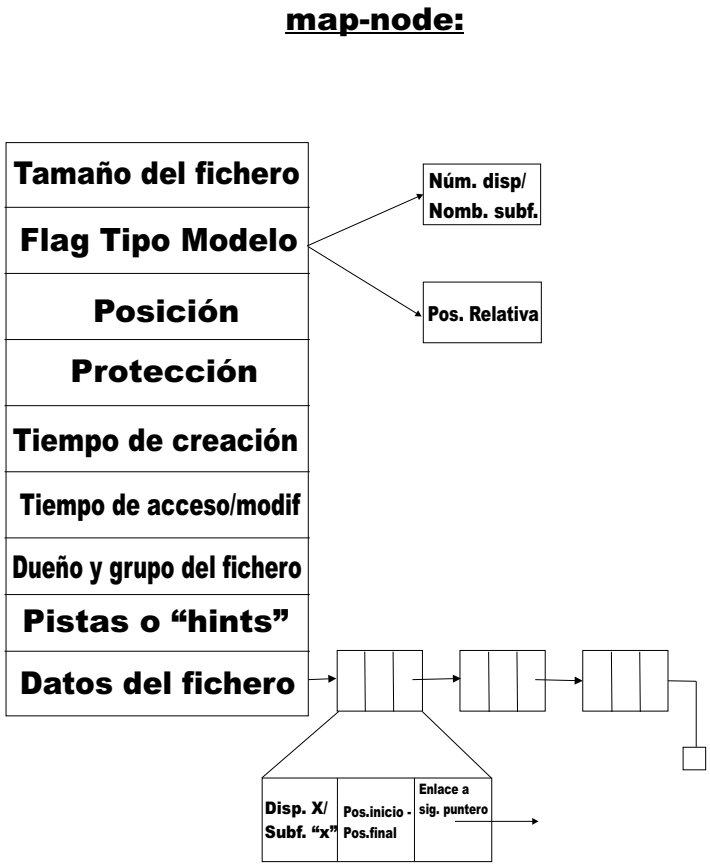


Figura 5.5: Estructura del *m*ap-node

5.4.2. Distribución de los datos de los ficheros

Para realizar la distribución de la información dentro del sistema de ficheros MAPFS, vamos a utilizar el concepto de grupo de almacenamiento, que se describe en el capítulo 7. En las secciones 7.4 y 7.4.2 se analiza la distribución de los datos y su funcionamiento interno.

5.4.3. Nombrado de ficheros

Un aspecto fundamental de cualquier sistema de ficheros es el mecanismo de nombrado de los ficheros. En MAPFS se va a utilizar un esquema de nombrado basado en los sistemas de ficheros tipo UNIX y se va a utilizar el concepto de directorio. En el caso de MAPFS hay que extender dicho concepto para que permita el manejo de grupos de almacenamiento. El nombrado de ficheros se describe en la sección 7.8.

5.5. Subsistema multiagente de MAPFS

MAPFS consiste en un sistema de ficheros paralelo integrado con un sistema multiagente, que se encarga fundamentalmente de la gestión de la recuperación de la información (*information retrieval*). Uno de los campos donde los agentes pueden ser muy útiles es precisamente en la construcción de un sistema de recuperación de información (*information recovery systems*). Como hemos visto an-

teriormente, MAPFS consta de dos subsistemas con objetivos claramente definidos: (i) el sistema MAPFS_FS, que proporciona las capacidades de un sistema de ficheros paralelo y (ii) MAPFS_MAS, que se encarga precisamente de la tarea de la recuperación de la información. MAPFS_MAS es un subsistema independiente, que proporciona soporte al subsistema principal (MAPFS_FS) en cuatro áreas diferentes:

- **Acceso a la información:** Esta característica es la tarea principal del sistema MAPFS_MAS y consiste en la recuperación de la información. Dicha información se encuentra almacenada en los nodos de E/S (concretamente, un conjunto de discos distribuidos a través de varios nodos). Los agentes interaccionan en última instancia con la interfaz del subsistema MAPFS_FS.
- **Servicio de *caching*:** MAPFS explota la localidad temporal y espacial de los datos almacenados en los servidores. Para ello, utiliza una cache, que contiene una copia de los datos más recientes en un dispositivo de almacenamiento más rápido que el dispositivo de almacenamiento original. Sin embargo, utilizar una cache implica un problema importante de coherencia. Dentro del subsistema MAPFS_MAS existen un conjunto de agentes que gestionan esta característica. Estos agentes se denominan **agentes cache** y son los responsables de utilizar un protocolo de coherencia de cache y de controlar la transferencia de información entre ambos dispositivos de almacenamiento. Para mejorar la eficiencia del sistema, los agentes cache pueden construir y utilizar *h*ints, característica que se describe en el capítulo 8.
- **Servicio de tolerancia a fallos:** La idea de utilizar la metodología de agentes para dotar al sistema de tolerancia a fallos radica en la idea de que dicha característica se gestione de forma distribuida e independiente del cliente MAPFS. Por este motivo, se utilizan agentes de tolerancia a fallos procedentes del subsistema MAPFS_MAS.
- **Uso de *hints*:** Como se analiza en el capítulo 8, el sistema multiagente puede crear crear *hints* de forma dinámica, mediante el análisis de los patrones de acceso en tiempo de ejecución y la posterior construcción de dichos *hints*.

A fin de lograr dichos objetivos y como hemos visto, el sistema MAPFS_MAS está formado por un conjunto de agentes que interaccionan entre ellos, es decir, un sistema multiagente (MAS). El uso de un MAS implica coordinación entre los agentes de los que consta.

5.5.1. Jerarquía de agentes

Los agentes utilizados para la resolución de un problema pueden asociarse a un determinado nivel de abstracción del mismo. De hecho, descomponer un problema en niveles de abstracción permite simplificar el proceso de descomposición de tareas [LE80], [WHRB+81].

MAPFS utiliza una jerarquía de agentes, que permite resolver de una forma eficiente y transparente el problema de la adquisición de los datos. Según las tareas que debe llevar a cabo el MAS, el sistema está formado por los siguientes tipos de agentes:

- **Agentes extractores:** Se encargan de llevar a cabo la adquisición de datos en el sistema de ficheros MAPFS.
- **Agentes distribuidores:** Se van a encargar de distribuir a los agentes extractores. Éstos constituyen un nivel más alto en la jerarquía de los agentes.
- **Agentes de cache:** Están asociados a uno o más agentes extractores en cada instante, existiendo la posibilidad de modificar este enlace. Se utilizan para proporcionar eficiencia al sistema, ya

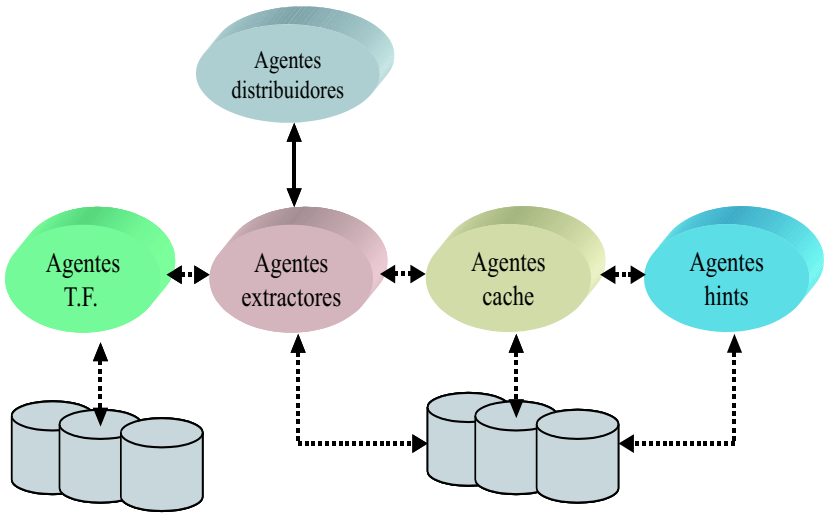


Figura 5.6: Relación entre los distintos tipos de agentes que forman el sistema

que permiten que la información se adquiera en un tiempo más corto, al guardar una copia de los datos en un soporte de almacenamiento que tiene una velocidad de acceso mayor.

- **Agentes de tolerancia a fallos:** Al sistema de adquisición de datos se le desea proporcionar características de tolerancia a fallos (visto desde el punto de vista de disponibilidad de datos). Es necesario introducir esta característica para evitar que un fallo de uno de los agentes afecte a todo el sistema. Para ello, se utilizarán los denominados **agentes de tolerancia**, que permiten proporcionar tolerancia al sistema de ficheros MAPFS.
- **Agentes constructores de *hints*:** Las tareas desarrollados por estos agentes están basadas en las optimizaciones descritas en el capítulo 8. No obstante, en este capítulo se describen las interacciones de estos agentes con el resto de la jerarquía.

En la figura 5.6 queda representada la relación entre las diferentes clases de agentes. A continuación, se describe el modelo de cooperación de los mismos.

5.5.2. Modelo de cooperación de MAPFS

Uno de los aspectos definitorios de un sistema multiagente es la cooperación entre los agentes que lo forman. Dichos agentes cooperan con el objetivo de resolver un problema global. En el caso de MAPFS, el problema global consiste en incrementar la eficiencia del acceso a los datos y proporcionarle robustez, añadiéndole la característica de tolerancia a fallos [PGC02]. Debido a la modularidad del paradigma de agentes, es posible ampliar el número de características del sistema multiagente, extendiendo la funcionalidad del mismo.

La coordinación entre diferentes agentes para la resolución de un problema implica la descomposición del problema global en problemas parciales, que son resueltos por cada uno de los agentes de una forma coordinada. Por tanto, es necesario llevar a cabo una representación de estos problemas parciales. De hecho, la descomposición de los problemas depende en gran parte de su formulación [BG88b]. De este modo, se han formalizado varios conceptos que nos permiten describir las tareas de los agentes y el modelo de cooperación del MAS [HS95]. Esta formalización se ha realizado en base a la visión **simbólica** de un agente.

- En primer lugar, cada agente debe conocer sus **metas**, es decir, la descripciones del estado deseado del entorno de los agentes. Las metas dependen del tipo de agente utilizado, a saber:

  - Los agentes extractores son subordinados de los agentes distribuidores y no dependen del entorno.
  - Las metas de los agentes distribuidores son totalmente dependientes del entorno y son las más parecidas a los objetivos del sistema completo. Estas metas corresponden a las peticiones de los usuarios.
  - Las metas de los agentes cache corresponden a las peticiones de los agentes extractores. Cada petición de información realizada por los agentes extractores, es resuelta por los agentes cache. Si la información no está disponible en la estructura cache, dichos agentes intentan obtener los datos.
  - Los agentes de tolerancia a fallos son activados sólamente cuando la copia original de los datos en los nodos no está disponible.
  - Los agentes *hints* se encargan de la construcción de dicha metainformación con el objetivo de incrementar el rendimiento de la E/S. Pueden activarse por parte de cualquiera de los agentes, dependiendo del nivel donde se aplique el uso de esta metainformación. En el caso de MAPFS, se ha limitado su uso por parte de los agentes extractores y los agentes cache, como quedó reflejado en la figura 5.6.

  Formalmente, las metas de los agentes pueden ser representadas y descritas de la siguiente forma:

  - g_{da}**:** metas de los agentes distribuidores;
  - g_{ea}**:** metas de los agentes extractores;
  - g_{ca}**:** metas de los agentes cache;
  - g_{fta}**:** metas de los agentes de tolerancia a fallos;
  - g_{ha}**:** metas de los agentes *h*ints.

  El modelo de cooperación se establece a nivel de grupo de almacenamiento. Aunque el capítulo 7 los describe de forma detallada, en esta sección es suficiente saber que un grupo de almacenamiento es una agrupación de servidores de almacenamiento en el sistema de ficheros MAPFS. Van a existir diferentes subsistemas multiagente, uno por cada grupo de almacenamiento. A continuación se definen las metas de los diferentes agentes:

  $g_{da}(x) = exists(d, G_y)$
  donde `x` es un agente distribuidor perteneciente a cualquier servidor
  $S_z / S_z \in G_y$
  $\wedge$ `d` es un objeto concreto
  $\wedge$ *exists* es un predicado que indica si un objeto está
  disponible para un usuario en un grupo de almacenamiento.

$g_{ea}(x) = serves(x, y)$
donde `x` es un agente extractor perteneciente a cualquier grupo de almacenamiento G_z/ `y` es un agente distribuidor perteneciente al mismo grupo
$\wedge$ *serves* es un predicado que indica si `x` ha satisfecho la petición del agente `y`.

$g_{ca}(x) = provides(x, y)$
donde `x` es un agente cache perteneciente a cualquier grupo de almacenamiento G_z/ `y` es un agente extractor perteneciente al mismo grupo
$\wedge$ *provides* es un predicado que indica si `x` tiene los objetos requeridos por el agente `y` en la estructura cache.

$g_{fta}(x) = provides\_fault\_tolerance(x, y)$
donde `x` es un agente de tolerancia a fallos perteneciente a cualquier grupo G_z/ `y` es un agente extractor perteneciente al mismo grupo
$\wedge$ *provides_fault_tolerance* es un predicado que es falso sólo cuando el agente `y` no puedo obtener los datos y el agente `x` no puede proporcionar tales datos. En otro caso, el predicado es verdadero.

$g_{ha}(x) = provides\_hints(x, y)$
donde `x` es un agente *hint* perteneciente a cualquier grupo G_z/ `y` es un agente cache perteneciente al mismo grupo
$\wedge$ *provides_hints* es un predicado que es falso sólo cuando el agente `y` no puedo obtener la metainformación solicitada por el agente `y`
En otro caso, el predicado es verdadero.

- De acuerdo a las metas de cada agente, los planes contienen instrucciones o acciones precisas para lograr alcanzar los objetivos. Nuevamente, las acciones y planes dependen del tipo de agente. Se definen de la siguiente forma:
  - p_{da}**:** planes de los agentes distribuidores
  - p_{ea}**:** planes de los agentes extractores

- p_{ca}**:** planes de los agentes cache
- p_{fta}**:** planes de los agentes de tolerancia a fallos
- p_{ha}**:** planes de los agentes *hints*

$p_{da}(x) = if \neg exists(d, G_y) \longrightarrow \forall y \ / \ is\_a\_ea(y)$
$\wedge$ y $\in G_y$ entonces *ask(d,y)*
donde `x` es un agente distribuidor perteneciente a cualquier servidor
$S_z / S_z \in G_y$
$\wedge$ `d` es un objeto concreto
$\wedge$ *is_a_ea* es un predicado que es verdadero si `y` es
un agente extractor y falso en caso contrario
$\wedge$ *ask* es una función que genera un evento que permite
solicitar que el agente `y` realice la recuperación del objeto `d`.

$p_{ea}(x) = if \neg serves(x, y) \wedge is\_asked(y, d) \longrightarrow$
$\forall z \ / \ is\_a\_ca(z)$ entonces *ask(d,z)*
donde `x` es un agente extractor perteneciente a cualquier grupo
de almacenamiento $G_z /$ `y` es un agente distribuidor perteneciente al
mismo grupo $\wedge$ `d` es un objeto concreto
$\wedge$ *is_a_ca* es un predicado que es verdadero si `z` es un agente cache
y falso en otro caso
$\wedge$ *ask* es una función que genera un evento que permite
solicitar que el agente `z` realice la recuperación del objeto `d`.

$p_{ca}(x) = if \neg provides(x, y) \wedge is\_asked(y, d) \longrightarrow$
$obtain(d)$
donde `x` es un agente cache perteneciente a cualquier grupo
de almacenamiento $G_z /$ `y` es un agente extractor perteneciente al
mismo grupo $\wedge$ *obtain* es una función que permite obtener la
información del disco y almacenarla en la estructura cache.

$p_{fta}(x) = if \neg provides\_fault\_tolerance(x, y)$
$\wedge is\_asked(y, d) \longrightarrow obtain\_duplicate(d)$
donde `x` es un agente de tolerancia a fallos perteneciente a cualquier
grupo de almacenamiento $G_z /$ `y` es un agente extractor perteneciente
al mismo grupo $\wedge$ *obtain_duplicate* es una función que permite obtener
la información de la réplica del disco y proporcionársela al agente `y`.

$p_{ha}(x) = if \neg provides\_hints(x, y)$
$\wedge\, is\_asked(y, h) \longrightarrow obtain(h)$
donde `x` es un agente *hint* perteneciente a cualquier grupo de almacenamiento G_z/ `y` es un agente extractor perteneciente al mismo grupo $\wedge$ *obtain* es una función que permite obtener la metainformación y proporcionársela al agente `y`.

Las funciones *obtain* y *obtain_duplicate()* deben proyectarse en el sistema MAPFS_FS como operaciones de lectura/escritura paralelas, que dependiendo de la aplicación concreta pueden ser no contiguas, colectivas o no bloqueantes.

- Cada agente debe contener un conjunto de **eventos** planificados, que deben gestionarse a través del uso de agentes. Hay dos clases de eventos: (i) los eventos originados por el usuario y (ii) los eventos originados por los propios agentes. El primer tipo es el origen del resto de eventos, ya que sólo cuando una aplicación de usuario realiza una petición de E/S, se inicia el **árbol de eventos**. Dicho árbol aparece en la figura 5.7.

  Como se puede observar en dicha figura, el orden en el cual se generan los eventos es el siguiente:

  1. En primer lugar, un programa de usuario realiza una petición de E/S. Esto genera un evento de usuario, que es capturado por un agente distribuidor.
  2. Esto genera a su vez un evento de extracción, dirigido hacia un agente extractor, el cual es el responsable de obtener y enviar los datos.
  3. Normalmente, el agente extractor busca los datos en la cache, delegando en el agente cache esta tarea. El agente cache debe llevar a cabo tanto la devolución de los datos al agente extractor como su almacenamiento en la estructura cache.
  4. A menos que los datos no estén disponibles en el disco original, el evento de tolerancia a fallos no se genera. En caso necesario, el agente de tolerancia a fallos es el encargado de obtener los datos del disco de soporte. Estos datos no son almacenados en la estructura cache, debido a que un evento de esta índole es poco probable.

- De acuerdo con la planificación, los agentes deben ejecutar el plan. El modelo de planificación es **dirigido por eventos**. Si se genera un evento y las premisas o condiciones se cumplen, las acciones correspondientes se ejecutan, modificándose el estado del sistema.

- La **cooperación** se logra a través del uso de **planes compartidos**, es decir, realizando la planificación de una forma compartida. De esta forma, la cooperación se alcanza a través de dos esquemas diferentes: (i) hay réplicas de todos los agentes; dichos agentes deben coordinar sus esfuerzos con el objetivo de satisfacer las metas globales del sistema; por ejemplo, la estructura cache debe dividirse en secciones y cada agente cache debe gestionar una sección; el agente distribuidor es el encargado de distribuir el trabajo. Esta clase de planificación es denominada **intra-planificación**. (ii) Cada grupo de almacenamiento debe interoperar con el resto de los grupos de almacenamiento a fin de planificar el sistema completo. Esta planificación es denominada **inter-planificación**.

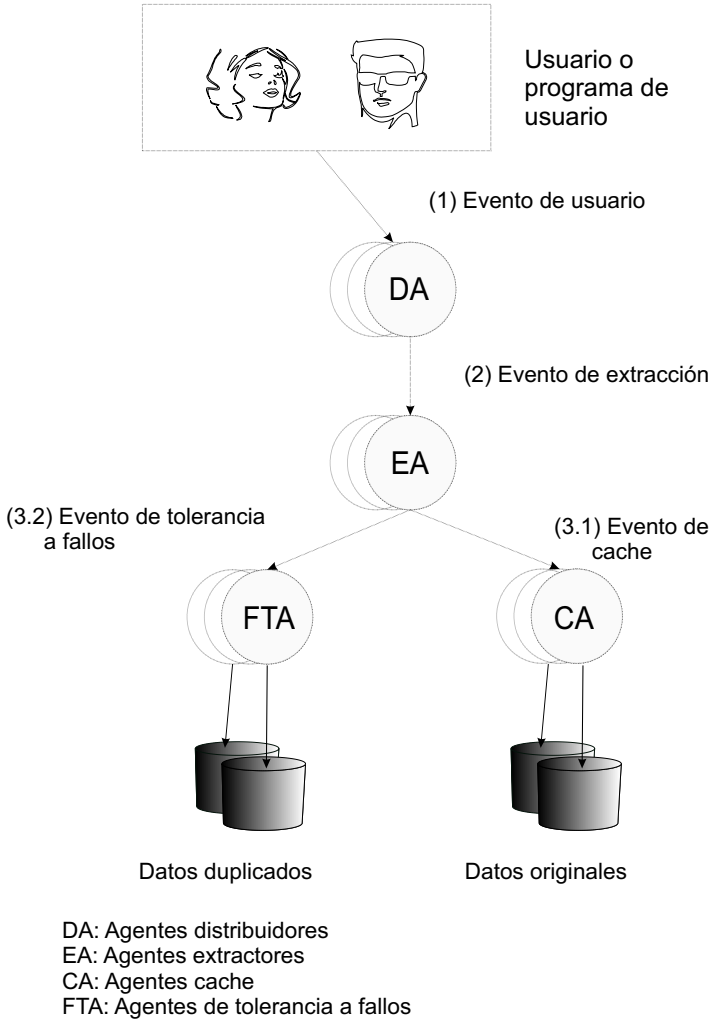


Figura 5.7: Árbol de eventos en MAPFS

5.5.3. Relación entre la arquitectura de MAPFS y el MAS

Esta sección describe cómo el sistema multiagente encaja dentro de la arquitectura MAPFS, analizando el contenido de cada uno de los módulos de MAPFS:

- Interfaz de Usuario: Este módulo proporciona accesso a la funcionalidad de MAPFS, ofreciendo la interfaz del sistema. Por tanto, el MAS no está relacionado con este módulo.

- Gestor de Comunicación: Este módulo es el responsable de tres tareas diferentes, que se describieron previamente: (i) transferencia de información en el sistema MAPFS; (ii) comunicación entre el Gestor de Cache y el Gestor de Ficheros y el servidor de almacenamiento a fin de que la cache pueda actualizarse y (iii) distribución de los agentes en el sistema MAPFS. Este módulo es el canal de distribución de los agentes pertenecientes a todos los MAS.

- Gestor de Cache: Es el módulo encargado de modelar el protocolo de coherencia de cache, que implementan los agentes cache.

- Gestor de Tolerancia a Fallos: Este módulo garantiza la disponibilidad de los datos. Esta característica la realizan los agentes de tolerancia a fallos.
- Gestor de Ficheros: Este módulo implementa la funcionalidad correspondiente al subsistema MAPFS_FS. Los agentes extractores realizan peticiones a este módulo.
- Agencia: Se trata del soporte software donde los diferentes agentes se crean y desarrollan su trabajo. Por tanto, el Gestor de Comunicación, el Gestor de Cache, el Gestor de Tolerancia a Fallos y el Gestor de Ficheros están enlazados con la agencia.

Las metas y planes deben implementarse en los módulos correspondientes, dependiendo del tipo de agente asociado a los mismos.

5.6. Resumen

Este capítulo ha descrito el diseño del sistema de ficheros MAPFS desde un punto de vista arquitectónico, haciendo hincapié en la división en subsistemas del sistema global; por un lado, el subsistema de ficheros, que se encarga de las funcionalidades relativas a la gestión y modelización del fichero y por otro lado, el subsistema multiagente, que se encarga de la adquisición de datos, y que además añade la funcionalidad de *caching* y tolerancia a fallos, dejando abierta la posibilidad de extender dicha funcionalidad.

Adicionalmente, el capítulo ha analizado el modelo de fichero utilizado en MAPFS, detallando la estructura que define el mismo y la interfaz de acceso al sistema de ficheros.

Capítulo 6

Estructura de un agente en el sistema MAPFS

6.1. Introducción

La tecnología de agentes permite abordar problemas de una forma completamente nueva en el mundo de la computación. A pesar de estar intimamente relacionado con el mundo de la Inteligencia Artificial Distribuida (IAD) [AG92], [BG88b], [LL93], [Lab96], [BG92] algunos trabajos han demostrado que esta disciplina puede utilizarse en campos totalmente ajenos al mismo. En la última década, un gran número de aplicaciones han aparecido en el campo de los negocios [JFN+96], en gestión eléctrica [JCL+95b], [CN96], en control [ADS99], [AFC00], en redes [GCS96] o aplicaciones industriales, en general [Häg97], [Jen92]. La principal ventaja de la tecnología de agentes es su aporte conceptual, que permite llevar a cabo el análisis y diseño de aplicaciones desde un punto de vista más cercano al lenguaje natural y dotar a las mismas de una serie de propiedades que les proporcionan una mayor capacidad para enfrentarse a entornos complejos y cambiantes.

Por otro lado, existen diferencias que parecen irreconciliables entre la denominada *programación de sistemas* y las soluciones propuestas dentro de la tecnología de agentes. Los principales motivos son:

- La programación de sistemas interacciona a muy bajo nivel con el propio sistema. El paradigma de agentes permite una interacción a un nivel mucho más alto.

- La eficiencia es un requisito muy estricto en el caso de la programación de sistemas. Introducir una capa de abstracción siempre implica una pérdida de eficiencia en la ejecución de cualquier sistema.

No obstante estas desventajas pueden ser salvadas, debido a que la disciplina de agentes diferencia claramente la teoría de agentes de las arquitecturas de agentes (véase secciones 3.5 y 3.6), siendo las primeras las que proporcionan los conceptos y las segundas las que proponen ya soluciones concretas.

Esta tesis muestra cómo utilizar la teoría de agentes como marco conceptual para el diseño y desarrollo de un sistema de ficheros, utilizando tecnologías más cercanas a la programación de sistemas, pero beneficiándose de la aportación semántica de los agentes.

6.2. Estructura genérica de un agente

El concepto de agente ofrece un conjunto de propiedades muy interesantes desde el punto de vista de la computación. Algunas de estas características son la autonomía, la reactividad y la proactividad, que facultan al sistema a adaptarse a cambios en las condiciones de ejecución. Según Jennings et al. en [JSW98], también habría que añadir como características propias de un agente la situación en la cual se encuentra y la flexibilidad.

Además, una característica adicional muy ligada a los agentes y muy interesante en el caso del sistema de ficheros MAPFS es la de la inteligencia. Los agentes inteligentes suelen realizar frecuentemente la toma de decisiones del sistema. En este contexto, el sistema de agentes de MAPFS se encarga de la construcción de *hints* de forma dinámica, modificando los mismos según el conocimiento que se vaya adquiriendo al analizar los patrones de información que se extraigan. Como se verá en el capítulo 8, los *hints* o pistas pueden ser definidos por el usuario o calculados por el subsistema multiagente. En este último caso, es necesario definir un algoritmo de procesamiento de los *hints*, que permita aprovechar esta información de una forma adecuada.

Por otro lado, existen diferentes tipos de agentes, los cuales se encargan de realizar diferentes tareas en el sistema, como se describe en la sección 5.5.1. Por tanto, es necesario identificar en el sistema el rol o papel de dicho agente. Esta característica ya ha sido identificada y utilizada en algunas arquitecturas de agentes, entre las que destaca la plataforma de agentes MADKIT [GF01]. Esta arquitectura define el modelo denominado AGR (*Agent-Group-Role*), en el cual el papel o labor de un agente constituye uno de los conceptos clave del mismo. El papel o rol consiste en una representación abstracta de la función o servicio que ofrece un agente. Del mismo modo, en MAPFS se utiliza el concepto de rol para distinguir la función específica de un agente.

Definición 1 *En MAPFS, un agente se define formalmente como la siguiente tupla:*

$$< \texttt{Id\_Ag}, \texttt{Rol}, \texttt{Red\_Int} >$$

donde:

- `Id_Ag`: *Se trata de un identificador del agente, que debe referenciar de forma unívoca a cada uno de los agentes que forman el sistema.*
- `Rol`: *Representa el tipo de agente, tomando valores en el siguiente dominio:* `[Cache, Distribuidor,Extractor,Tolerancia, Hint]`. *No obstante, este dominio puede incrementarse con otros valores, siempre que se implemente el servicio correspondiente.*
- `Red_Int`: *Representa la red de interacción del agente específico con otros agentes que se comunican con el mismo. La red de interacción puede representarse como un vector de las relaciones existentes entre el agente* `Id_Ag` *y el resto de los agentes, de modo análogo a la construcción de la matriz de agrupación que se verá en la sección 7.3.2.*

En adición a estos campos, aparece el concepto de grupo de almacenamiento, como se describe en el capítulo 7. Un agente queda asociado a uno de estos grupos, con el objetivo de atender sus peticiones. Por razones de simplicidad, un agente sólo se utiliza para un único grupo de almacenamiento. Añadiendo esta característica, la definición de grupo de almacenamiento queda modificada como se muestra a continuación.

Definición 2 *Considerando el concepto de grupo de almacenamiento, un agente en MAPFS se define como la tupla:*

$$< \texttt{Id\_Ag}, \texttt{Grupo}, \texttt{Rol}, \texttt{Red\_Int} > \tag{6.1}$$

donde `Grupo` *identifica el grupo de almacenamiento al cual pertenece el agente y* `Red_Int` *consiste en el vector de relaciones entre el agente* `Id_Ag` *y el resto de agentes pertenecientes al grupo de almacenamiento* `Grupo`*.*

Uno de los aspectos fundamentales de un sistema multiagente es la comunicación entre los diferentes agentes que forman parte del mismo. Como se describe en el capítulo 3, existen lenguajes específicos para agentes que permiten dicha comunicación, entre los que destaca el lenguaje KQML. Este lenguaje está formado por un conjunto de mensajes, denominados *performatives*, que permiten de una forma flexible especificar elementos de comunicación entre los agentes. En el artículo [LF97a], Labrou y Finin describen de forma detallada las *performatives* reservadas, algunas de las cuales se utilizan en esta sección.

A partir del modelo de cooperación de MAPFS, se ha definido el conjunto de *performatives* que permiten la comunicación entre los agentes pertenecientes al MAS de un determinado grupo de almacenamiento. Previo a la definición de las *performatives*, se necesita definir los siguientes conjuntos de elementos para un determinado grupo de almacenamiento:

DA: Conjunto de agentes distribuidores

EA: Conjunto de agentes extractores

CA: Conjunto de agentes cache

FTA: Conjunto de agentes de tolerancia a fallos

HA: Conjunto de agentes *hints*

A continuación se describen las *performatives* KQML necesarias para la interacción de los agentes del sistema.

Cuando se realiza una petición de un elemento `d`, el agente distribuidor se encarga de realizar las peticiones del mismo a los diferentes agentes extractores. Supongamos que `x` es un agente distribuidor de un grupo de almacenamiento G_x. La figura 6.1 contiene la *performative* que lleva a cabo el agente distribuidor.

Si el agente extractor seleccionado posee el elemento `d`, entonces dicho agente lleva a cabo la *performative* de la figura 6.2, indicándole al agente distribuidor que el dato `d` está disponible en el grupo de almacenamiento G_x.

Por otro lado, si el agente extractor no posee el elemento `d`, es decir, no se encuentra en la estructura cache, debe ejecutar la *performative* de la figura 6.3, solicitando a cada agente cache la obtención de dicho dato.

El predicado `ask(d,z)` en el agente cache `z` provoca la ejecución de la función `obtain(d)`.

A continuación, el agente cache envía al agente distribuidor, a través del agente extractor asociado al primero, información sobre la finalización de la operación, indicándole que el elemento `d` se encuentra disponible en el grupo de almacenamiento G_x. Este proceso se lleva a cabo a través de la *performative* de la figura 6.4.

De este modo, se cierra el ciclo. No obstante, la estructura cache tiene un máximo conjunto de entradas, que deben reemplazarse por otros elementos a través de una determinada política de

Paso 1

$x \in DA$

$y \in EA$

(ask-if

:sender x

:receiver y

:reply-with id_da

:language Prolog

:ontology MAPFS

:content "`exists(d,`G_x`)`")

Figura 6.1: *P*erformative de petición de datos de un agente distribuidor a un agente extractor

Paso 2.1

(tell

:sender y

:receiver x

:in-reply-to id_da

:reply-with id_ea

:language Prolog

:ontology MAPFS

:content "`exists(d,`G_x`)`")

Figura 6.2: *P*erformative de respuesta de un agente extractor a un agente distribuidor si posee los datos solicitados

reemplazo. Cuando se invalida una determinada entrada, el agente cache `z` envía la *performative* representada en la figura 6.5.

El agente cache también puede hacer uso de los metadatos proporcionados por los agentes *hints*, enviando la *performative* de la figura 6.7. De este modo, se solicita la metainformación identificada por h.

El agente *hint* se encarga de la construcción de la metainformación necesaria, devolviéndosela al agente cache a través de la *performative* de la figura 6.8.

Por otro lado, si se proporciona tolerancia a fallos al sistema, es necesario que cada agente extractor contacte con un determinado agente de tolerancia a fallos de forma simultánea a la petición de datos al agente cache. Por tanto, en el agente extractor y se ejecutará la *performative* de la figura 6.6.

En este caso, el predicado `ask(d,v)` en el agente de tolerancia a fallos `v` provoca la ejecución de la función `obtain_duplicate(d)`.

Como en el caso del agente cache, el agente de tolerancia a fallos envía al agente distribuidor, a través del agente extractor asociado, un aviso sobre la finalización de la operación. De este modo, se

Paso 2.2

$z \in CA$

(achieve

:sender y

:receiver z

:in-reply-to id_da

:reply-with id_ea

:language Prolog

:ontology MAPFS

:content "`ask(d,z)`")

Figura 6.3: *P*erformative de petición de datos de un agente extractor a un agente cache

Paso 3.1

(forward

:from z

:to x

:sender z

:receiver y

:reply-with id_ca

:language KQML

:ontology kqml-ontology

:content (achieve

:sender z

:receiver x

:in-reply-to id_ea

:reply-with id_ca

:language Prolog

:ontology MAPFS

:content "`exists(d,`G_x`)`")

Figura 6.4: *P*erformative de respuesta de un agente cache a un agente distribuidor una vez obtenidos los datos

indica al agente distribuidor que el elemento `d` se encuentra disponible en el grupo de almacenamiento G_x, debido a la disponibilidad de los datos, proporcionada por los agentes de tolerancia a fallos. Este proceso se lleva a cabo a través de la *performative* de la figura 6.9.

Como se puede comprobar, los pasos 3.1 y 3.3 producen el mismo resultado, siendo redundante la

Invalidación

(forward
:from z
:to x
:sender z
:receiver y
:reply-with id_ca'
:language KQML
:ontology kqml-ontology
:content (unachieve
:sender z
:receiver x
:in-reply-to id_ea
:reply-with id_ca'
:language Prolog
:ontology MAPFS
:content "`exists(d,`G_x`)`")

Figura 6.5: *P*erformative de invalidación de un determinado dato en la cache

Paso 2.3

$v \in FTA$

(achieve
:sender y
:receiver v
:in-reply-to id_da
:reply-with id_ea'
:language Prolog
:ontology MAPFS
:content "`ask(d,v)`")

Figura 6.6: *P*erformative de petición de datos de un agente extractor a un agente de tolerancia a fallos

ejecución de ambos pasos. Precisamente esa redundancia es la que se desea conseguir en el sistema, a través del uso de los agentes de tolerancia a fallos.

La figura 6.10 muestra el flujo de ejecución de las *performatives* del sistema. En línea discontinua se representa el escenario en el cual el dato solicitado está en la estructura cache y en línea punteada se representan escenarios más complejos, tales como aquél en el que los datos solicitados no se encuen-

Paso 3.2
$u \in HA$
(achieve
:sender z
:receiver u
:reply-with id_ca"
:language Prolog
:ontology MAPFS
:content "`ask(h,v)`")

Figura 6.7: *P*erformative de petición de *h*ints de un agente extractor a un agente *h*int

Paso 4.1
(tell
:sender u
:receiver z
:in-reply-to id_ca"
:reply-with id_ha
:language Prolog
:ontology MAPFS
:content "`exists(h,`G_x`)`")

Figura 6.8: *P*erformative de respuesta de un agente *h*int a un agente cache una vez obtenidos los *h*ints correspondientes

tran disponibles en la estructura cache y es necesario acceder al dispositivo de almacenamiento para obtenerlos, o bien, cuando se desea proporcionar tolerancia a fallos o finalmente, cuando se utiliza el agente *hint* para mejorar el proceso de *caching/prefetching*.

6.3. Agente *proxy* o cache. Un ejemplo de agente

Como ejemplo de agente en MAPFS se ha utilizado lo que a lo largo del texto se ha denominado agente cache o agente *proxy*. Este último término es más adecuado, dado que en el presente trabajo no se consideran aspectos de coherencia, característica intrínsecamente relacionada con el concepto de cache. Por tanto, los agentes utilizados en dicha propuesta ejercen de *proxies* o elementos intermediarios que contienen una copia de los datos que se desea leer o escribir. Esta propuesta es suficiente para validar nuestro sistema, ya que lo que se desea comprobar dentro de este apartado de la tesis es que el paradigma de agentes se adapta adecuadamente a nuestro propósito de optimizar el rendimiento de un sistema de ficheros paralelo.

Por tanto, aunque de forma conceptual se habla de agente cache, la implementación realizada y

Paso 3.3

(forward
:from v
:to x
:sender v
:receiver y
:reply-with id_fta
:language KQML
:ontology kqml-ontology
:content (achieve
:sender v
:receiver x
:in-reply-to id_ea'
:reply-with id_fta
:language Prolog
:ontology MAPFS
:content "`exists(d,`G_x`)`")

Figura 6.9: *P*erformative de respuesta de un agente de tolerancia a fallos a un agente distribuidor una vez obtenidos los datos

evaluada está más cercana al concepto de agente *proxy*. No obstante, a lo largo del texto se nombrarán ambos términos indistintamente.

Una vez realizado este inciso, vamos a proceder a definir un agente cache. En primer lugar, es necesario detallar cuáles son los objetivos de este tipo de agente. Si analizamos el capítulo 5, se puede concluir que los agentes cache:

1. Están asociados a uno o más agentes extractores en cada instante.
2. Se utilizan para proporcionar eficiencia al sistema, ya que permiten que la información se adquiera en un tiempo más corto, al guardar una copia de los datos en un soporte de almacenamiento que tiene una velocidad de acceso mayor.
3. Son los responsables de utilizar un protocolo de coherencia de cache así como de controlar la transferencia de datos entre los dos dispositivos de almacenamiento, el disco del servidor y la memoria del cliente.

Debido a que la coherencia queda fuera de los objetivos de esta tesis, el cometido de los agentes cache se puede reducir a dos tareas ampliamente utilizadas en los sistemas de ficheros existentes, a saber, la labor de *prefetching* y de *caching* de los datos. La tarea de *prefetching* permite incrementar el rendimiento de las operaciones de lectura, debido a la lectura adelantada de los datos más probablemente utilizados en posteriores operaciones. La tarea de *caching* permite incrementar el rendimiento de las operaciones de lectura y escritura, debido a la localidad de los datos y a la posibilidad de retardar la escritura.

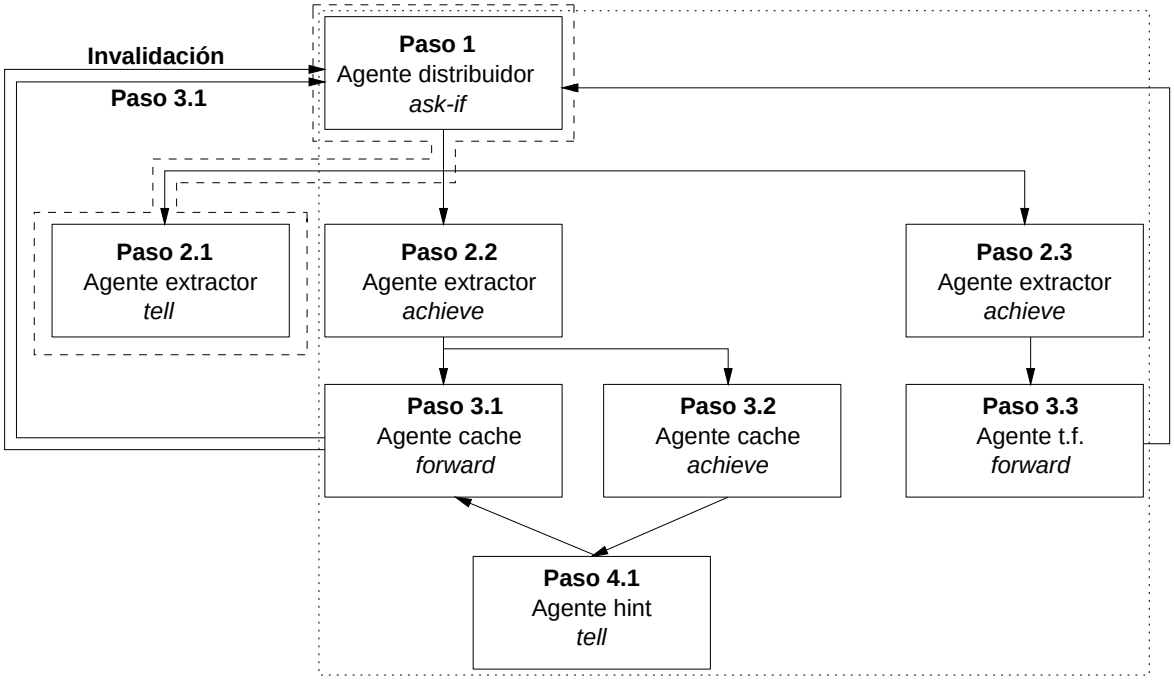


Figura 6.10: Flujo de ejecución de las performatives del sistema

A continuación se va a definir un agente cache, basándonos en la definición 6.1 de agente genérico.

Definición 3 *Un agente cache para un grupo de almacenamiento G_x se define como la tupla:*

$$< \texttt{Id\_Ag\_C}, \texttt{G}_x, \texttt{Cache}, \texttt{Red\_Int\_C} >$$

donde `Red_Int_C` *representa la red de interacción del agente cache con otros agentes que se comunican con el mismo. Se representa mediante un vector de relaciones entre el agente cache y el resto de agentes del grupo de almacenamiento* $\texttt{G}_x$*. Según la figura 5.6, se comprueba que los agentes cache están relacionados entre sí, con los agentes extractores y con los agentes hints. Por este motivo, el vector* `Red_Int_C` *cumple la siguiente expresión:*

$$Red\_Int\_C = (ri_i) \ donde \ ri_i = \begin{cases} 1 & \Longrightarrow i \in CA \parallel i \in EA \parallel i \in HA \\ 0 & \Longleftarrow i \in DA \parallel i \in FTA \end{cases}$$

Las dos condiciones de la anterior expresión son suficientes o necesarias, pero no ambas. La libertad ofrecida por la anterior expresión permite establecer diferentes relaciones de los agentes cache con los agentes extractores, *hint* y cache de su grupo de almacenamiento.

La relación existente entre un agente cache y un agente extractor se deriva de forma directa de la *performative* representada en la figura 6.3. De este modo, un agente cache al menos está asociado a un agente extractor. Además puede existir relación entre agentes cache si colaboran con el objetivo de servir a un único agente extractor. Los agentes cache también pueden relacionarse con agentes *hints*, a partir del uso de la directiva de la figura 6.7.

La *performative* de la figura 6.3 puede tener diferentes instancias, que corresponden a diferentes agentes colaboradores. Asimismo, pueden existir también diferentes instancias de la *performative* de la figura 6.7, si el agente cache colabora con diferentes agentes *hint*.

Los agentes cache se pueden clasificar de acuerdo al número de relaciones que se establecen entre ellos.

Definición 4 *Se denomina* **agente cache máximo** *a aquel agente cache relacionado con el máximo número de agentes posibles, es decir, aquel agente cache:*

$$< \texttt{Id\_Ag\_C}, \texttt{G}_x, \texttt{Cache}, \texttt{Red\_Int\_C} >$$

cuya red de interacción `Red_Int_C` *se define:*

$$Red\_Int\_C = (ri_i) \ donde \ ri_i = \begin{cases} 1 & \iff i \in CA \parallel i \in EA \parallel i \in HA \\ 0 & \iff i \in DA \parallel i \in FTA \end{cases}$$

Definición 5 *Se denomina* **agente cache mínimo** *a aquel agente cache relacionado con el mínimo número de agentes posibles. Como se mencionó anteriormente y por su semántica, un agente cache está asociado al menos a un agente extractor. Por tanto, un agente cache es un agente cache mínimo:*

$$< \texttt{Id\_Ag\_C}, \texttt{G}_x, \texttt{Cache}, \texttt{Red\_Int\_C} >$$

sii $\exists$ *un único agente con identificador* j, $j \in EA$, *tal que:*

$$Red\_Int\_C = (ri_i) \ donde \ ri_i = \begin{cases} 1 & \iff i = j \\ 0 & e.o.c. \end{cases}$$

Como hemos mencionado anteriormente, las tareas que debe llevar a cabo un agente cache son la de *caching* y *prefetching* de datos. Para cada una de estas tareas, existe una característica configurable que permite establecer el tamaño o la ventana de datos que se van a procesar. A este parámetro se le denomina factor y se define a continuación.

Definición 6 *Se denomina* **factor de *c*aching** *al par* `[desplazamiento, numero_bloques]`, *que describe la ventana de datos que se va a guardar en cache después de una operación de lectura o escritura.*

Definición 7 *Se denomina* **factor de *p*refetching** *al par* `[desplazamiento, numero_bloques]`, *que describe la ventana de datos que se va a leer por adelantado y guardar en la estructura cache para posteriores accesos. La operación de prefetching puede realizarse junto a una operación de lectura, permitiendo traer datos por adelantado.*

Ambos factores son dependientes del tamaño de acceso. Los agentes cache tienen como objetivo calcular factores de *caching* y *prefetching* óptimos, o al menos, muy cercanos al óptimo, con la ayuda de los agentes *hints*.

A continuación vamos a ver como encaja el agente cache con las propiedades de un agente genérico, descritas anteriormente:

- Situación: el entorno en el que se encuentra un determinado agente cache corresponde al grupo de almacenamiento al cual pertenece. En cierto modo, el agente cache está confinado dentro de dicho grupo.
- Autonomía: Los agentes cache dependen de los agentes extractores, debido a que están asociados a los mismos, sirviéndolos como intermediarios. No obstante, los procesos de *prefetching* y *caching* pueden realizarse de forma autónoma, a partir de una petición del agente extractor.
- Reactividad: Los agentes deben ser reactivos a su entorno y a los estímulos producidos en el mismo. En este escenario, los estímulos corresponden a peticiones de E/S de las aplicaciones y a fallos del sistema, respecto a la disponibilidad de los datos. Las peticiones son directamente

recogidas por el agente distribuidor, que se comunica a través de *performatives* con los agentes extractores, los cuales a su vez se comunican por medio de *performatives* con los agentes cache. Por tanto, la comunicación entre agentes permite que un agente cache reaccione a estímulos producidos en el entorno de la aplicación. Por otro lado, si, por algún motivo, no se puede acceder a los datos, los agentes de tolerancia a fallos deben encargarse de resolver la problemática asociada.

- Proactividad y flexibilidad: Los agentes cache pueden determinar factores de *caching* y *prefetching* de una forma dinámica. Para ello, se pueden basar en ejecuciones pasadas, que se pueden almacenar en históricos clasificados por tamaño de acceso. Toda la información que permite mejorar estos aspectos se almacenan en forma de *hints*. Los agentes cache delegan en los agentes *hints* la recuperación o construcción de estos metadatos.
- Comunicación: Las *performatives* descritas en la sección 6.2 establecen el mecanismo de comunicación de todos los agentes entre sí, incluidos los agentes cache.

6.4. Resumen

En este capítulo se ha mostrado cómo se adapta el paradigma de agentes a un campo inicialmente ajeno al mismo, como es la E/S. En la introducción del capítulo se han referenciado algunos ejemplos de cooperación entre la disciplina de agentes y otras áreas de conocimiento, siendo satisfactorio el resultado de la aplicación.

A fin de alcanzar el objetivo propuesto, se describe la estructura genérica de un agente en el sistema MAPFS, realizando una definición formal de un agente y del modelo de comunicación utilizado entre los agentes que forman parte del sistema. Esta última parte es resuelta a través de la definición de *performatives* KQML adaptadas al dominio del problema.

Finalmente, se analiza más en detalle uno de los agentes utilizados en MAPFS, el agente cache, manifestando cómo las propiedades del mismo se adaptan a las propiedades genéricas de un agente.

Capítulo 7

Grupos de almacenamiento

7.1. Introducción

El concepto de agrupación es fundamental en todos los ámbitos de la vida. Edwin P. Hubble, al que se considera el fundador de la cosmología observacional, decía en los años 30 que el mejor sitio para buscar una galaxia era al lado de otra, afirmando una de las verdades fundamentales de la cosmología moderna: las galaxias se encuentran en agrupaciones. De hecho, toda la materia tiende a agruparse: las partículas subatómicas en átomos, los átomos en moléculas, las moléculas en objetos. Incluso los seres humanos nos agrupamos en lo que denominamos sociedad.

Al igual que el mundo real, el mundo de la informática también se encuentra formado por un gran número de ejemplos de agrupaciones. Conceptos tales como grupo de procesos, grupo de bloques o grupo de usuarios son utilizados para representar conjuntos de objetos o entidades propias de la computación.

La mayoría de estas agrupaciones se caracterizan por poseer la característica de sinergia, que de una forma sencilla significa que "el todo es más que la suma de las partes", es decir, que cuando dos o más componentes se agrupan, pueden originar resultados cuyas propiedades son diferentes a la suma de las propiedades individuales de cada componente. Esto es básicamente lo que se requiere a cualquier agrupación en informática: la utilidad del grupo frente a las características individuales de las entidades que constituyen el mismo.

7.2. Grupos de almacenamiento

Un **grupo de almacenamiento** se define como un conjunto de servidores que se agrupan como soporte de almacenamiento en un sistema de ficheros y, más concretamente dentro de este trabajo, el sistema de ficheros MAPFS.

Se dice que un fichero está asociado a un grupo de almacenamiento si dicho fichero se encuentra distribuido a través de los servidores que forman dicho grupo.

La formación de los grupos de almacenamiento se denomina **agrupación** y puede realizarse según

diferentes políticas, con el objetivo de optimizar el acceso por grupo de almacenamiento. De esta forma, se logrará alcanzar el objetivo final, que consiste en la optimización del acceso a los ficheros.

El concepto de grupo de almacenamiento es similar al de grupo de procesos, pero mientras el proceso es un concepto clave del sistema operativo, un grupo de almacenamiento queda enfocado dentro del sistema de ficheros.

De forma intuitiva, un grupo de almacenamiento parece tener un carácter más estático que un grupo de procesos, debido a que:

1. Un proceso es una entidad más dinámica y cambiante que un servidor de almacenamiento.

2. Un servidor de almacenamiento posee mayor complejidad que un proceso.

No obstante, uno de los objetivos de la definición de los grupos de almacenamiento es proporcionar dinamismo a la gestión de los servidores, de forma que el sistema MAPFS proporcione una interfaz que permita agregar y modificar de forma dinámica servidores a grupos de almacenamiento existentes o nuevos.

Por otro lado, del mismo modo que la partición es una abstracción lógica del concepto físico de disco, así el concepto de grupo de almacenamiento se puede ver como una abstracción lógica del concepto de servidor de almacenamiento[1].

Otro ejemplo que permite ver el concepto de grupo de almacenamiento como un paso más en la abstracción del sistema de E/S lo constituye el sistema de ficheros FFS (*Fast File System*) de UNIX BSD [MJLF84]. Dicho sistema de ficheros divide las particiones en grupos de bloques, los cuales contienen una copia del superbloque, así como de los nodos-i y los bloques de datos. En este caso, los motivos que llevan a usar los grupos de bloques son los siguientes:

1. Lograr que los bloques de datos se encuentren cerca de los nodos-i correspondientes.

2. Lograr que los nodos-i de los ficheros se encuentren cerca del nodo-i del directorio que los alberga.

De esta forma, se incrementa la velocidad de acceso a los datos.

En el caso del sistema de ficheros FFS, la agrupación se lleva a cabo dentro del propio sistema de ficheros y a nivel de partición. En el caso de los grupos de almacenamiento de MAPFS, por el contrario, la agrupación es realizada a nivel de conjunto de servidores, lo que constituye un salto en la jerarquía del sistema de E/S.

Teniendo en cuenta esto, las principales ventajas de los grupos de almacenamiento son las siguientes:

1. **Abstracción lógica del concepto de servidor:** Utilizar abstracciones lógicas de conceptos físicos en un sistema siempre facilita su gestión y explotación, así como permite modelar el sistema de cara a llevar a cabo su validación.

2. **Gestión dinámica de los servidores:** Como hemos indicado previamente, el uso de los grupos de almacenamiento permite una gestión dinámica de los servidores, a través de la interfaz ofrecida por el sistema MAPFS.

3. **Eficiencia de las operaciones de almacenamiento:** Del mismo modo que definir grupos de procesos permite que las operaciones sobre dichos grupos se realicen de forma más eficiente, también la definición de los grupos de almacenamientos mejora las operaciones sobre los mismos.

[1] Para ser exactos, es el concepto de grupo de particiones (en algunos sistemas como Windows 2000 llamados volúmenes seccionados o distribuidos) el que tiene esta semejanza con el grupo de almacenamiento

4. **Reparto de carga:** Establecer los grupos de almacenamiento, de forma que se ajusten los factores correspondientes[2] facilita y mejora el reparto de carga. Lo importante es conocer los valores correspondientes que optimizan esta característica.

5. **Migración transparente:** Un grupo de almacenamiento engloba a un conjunto de servidores y lo hace de forma transparente al usuario. El sistema puede ser capaz de llevar a cabo una modificación del conjunto de grupos de almacenamiento de forma transparente, a fin de mejorar y optimizar el funcionamiento del sistema completo.

Por otro lado, la única desventaja de la introducción de los grupos de almacenamiento en el sistema MAPFS es el incremento de la complejidad del mismo.

7.3. Características de los grupos de almacenamiento

En un primer paso, los grupos de almacenamiento pueden considerarse como una **partición** del conjunto de servidores que forman el sistema.

Para definir dicha partición, previamente debemos conocer algunas definiciones de la teoría de conjuntos [Hal87] (véase apéndice B).

En el caso de MAPFS, supongamos que tenemos un conjunto de servidores $\mathbf{S} = \bigcup_{i,j} S_i(j)$. Sobre dicho conjunto, definimos $\mathbf{G} = \{G_1, G_2, \dots G_n\}$ grupos de almacenamiento. Cada grupo G_i contiene un conjunto de servidores $S_{i(j)}$, es decir:

$$\begin{aligned} G_1 &= \{S_{1(1)}, \dots, S_{1(m)}\} \\ G_2 &= \{S_{2(1)}, \dots, S_{2(r)}\} \\ &\vdots \\ G_n &= \{S_{n(1)}, \dots, S_{n(l)}\} \end{aligned} \tag{7.1}$$

La aplicación que permite construir esta asociación entre servidores y grupos de almacenamiento se denomina **función de agrupación** y se denota como $\lambda_G : \mathbf{S} \rightarrow \mathbf{G}$. Se cumple que:

$$\lambda_G(S_i) = G_j \Longleftrightarrow S_i \in G_j$$

λ_G debe estar definida para todo el dominio de $\mathbf{S}$.

Sea R_G una relación definida sobre el producto cartesiano de los elementos de $\mathbf{S}$, que llamaremos **relación de agrupación**. La relación R_G entre dos servidores S_i y S_j se define como:

$$S_i R_G S_j \Longleftrightarrow \exists t / S_i \in G_t \wedge S_j \in G_t$$

Proposición 1 *Si la función de agrupación es una aplicación inyectiva, es decir, cada servidor sólo pertenece a un grupo de almacenamiento, se cumple la siguiente propiedad:*

$$S_i R_G S_j \Longleftrightarrow \lambda_G(S_i) = \lambda_G(S_j)$$

Demostración 1

$$S_i R_G S_j \Longleftrightarrow \exists t / S_i \in G_t \wedge S_j \in G_t \Longleftrightarrow \exists t / \lambda_G(S_i) = G_t = \lambda_G(S_j)$$

y además t es única, por ser la función de agrupación una aplicación inyectiva.

[2] realizar un *tuning*

Proposición 2 *Si la función de agrupación es una aplicación inyectiva, la relación R_G es una relación de equivalencia, que define la partición que aparece en la ecuación (7.1), es decir, R_G es una partición del conjunto de servidores* **S**. *Por tanto, cumple las propiedades reflexiva, conmutativa y transitiva.*

Demostración 2 *Véamos la demostración de que se cumplen las tres propiedades:*

- **Propiedad reflexiva:**

$$R_G \text{ es reflexiva sii } S_i R_G S_i \ \forall S_i \in \mathbf{S} \Longleftrightarrow \exists t / S_i \in G_t \wedge S_i \in G_t \Longleftrightarrow \exists t / S_i \in G_t$$
$$\Longleftrightarrow \exists t / \lambda_G(S_i) = G_t \tag{7.2}$$

Por definición de la función de agrupación, un servidor siempre está asociado a un grupo de almacenamiento al menos. Y en este caso, como la función de agrupación es inyectiva, sólo estará ligado a uno de ellos.

Luego la relación de agrupación cumple la propiedad reflexiva, c.q.d.

- **Propiedad simétrica:**

$$R_G \text{ es simétrica sii } (S_i R_G S_j \iff S_j R_G S_i) \text{ sii } ((\exists t / S_i \in G_t \wedge S_j \in G_t) \Longleftrightarrow (\exists l / S_j \in G_l \wedge S_x \in G_l)) \tag{7.3}$$

Por la naturaleza de la relación de agrupación, siempre va a darse esta propiedad. Eso se demuestra haciendo t=l en la anterior expresión.

Por tanto, R_G es simétrica c.q.d.

- **Propiedad transitiva:**

$$R_G \text{ es transitiva sii } S_x R_G S_y \wedge S_y R_G S_z \Longrightarrow S_x R_G S_z \text{ sii}$$
$$((\exists t / S_x \in G_t \wedge S_y \in G_t) \wedge (\exists l / S_y \in G_l \wedge S_z \in G_l)) \Longleftrightarrow (\exists p / S_y \in G_p \wedge S_x \in G_p)$$

Dado que la función de agrupación es inyectiva:

$$S_x R_G S_y \Longrightarrow \lambda_G(S_x) = G_u = \lambda_G(G_y)$$
$$G_y R_G G_z \Longrightarrow \lambda_G(S_y) = G_u = \lambda_G(S_z)$$
$$\Longrightarrow \lambda_G(S_x) = \lambda_G(S_z) \Longrightarrow G_x R_G G_z$$

Por tanto, R_G es transitiva c.q.d.

7.3.1. Limitaciones de la definición del grupo de almacenamiento como partición

La ventaja de utilizar una partición para definir los grupos de almacenamiento frente a otro tipo de agrupación diferente es la simplicidad de este tipo de relación. Esto puede verse simplemente analizando algunos escenarios:

- Si un servidor pertenece a dos grupos de almacenamiento diferentes, no se puede realizar de forma tan sencilla el equilibrado de carga: si un determinado grupo de almacenamiento es el de menor carga del sistema y se decide almacenar un nuevo fichero en dicho grupo por ese motivo, podría ocurrir que se estuviera sobrecargando otro grupo al cual pertenezca el servidor que comparte ambos grupos de almacenamiento.

- El cálculo y optimización de los parámetros de un grupo de almacenamiento, que veremos más adelante, se complica en el caso de que haya interrelaciones entre diferentes grupos de almacenamiento. Si la estructura de los grupos de almacenamiento se deja abierta, dicha optimización podría ser un *problema np-completo*.

No obstante, el problema de este tipo de grupos de almacenamiento es que no representan de forma correcta el dinamismo del sistema; es decir, no son adecuados para caracterizar situaciones en las cuales distintos servidores pueden unirse a grupos existentes o cambiar de grupo. ¿Qué ocurriría en ese caso con los ficheros que estaban almacenados en ese servidor? Por tanto, para estos casos veremos que es necesario ampliar el modelo utilizando otro tipo de agrupaciones para la construcción de la relación de agrupación.

Los grupos de almacenamiento que no cumplen las propiedades reflexiva, simétrica y transitiva los denotaremos como **grupos de almacenamiento no ortogonales** por contraposición al otro tipo de grupos de almacenamiento, que se denominan **grupos ortogonales**.

Para poder identificar las relaciones de agrupación óptimas, debemos previamente conocer cuál es el modelo de distribución de los datos en un grupo de almacenamiento, ya que la definición de dicho grupo está intimamente ligada al modelo de fichero utilizado en el sistema.

7.3.2. Representación de la relación de agrupación

La matriz de la relación de agrupación se denomina **matriz de agrupación** y se construye a partir de la siguiente expresión:

$$M_G = (mg_{ij}) \text{donde } mg_{ij} = \begin{cases} 1 & \text{si } (S_i, S_j) \in R_G \\ 0 & \text{si } (S_i, S_j) \notin R_G \end{cases} \tag{7.4}$$

En el caso de grupos ortogonales, se trata de una matriz simétrica, por lo que sólo es necesario almacenar la mitad de los elementos de esta matriz. Además, en este caso, se dice que M_G representa un cierre transitivo.

7.4. Modelo de distribución de los datos en MAPFS

Antes de pasar a definir el modelo de distribución de los datos en el sistema de ficheros MAPFS, planteémonos previamente por qué los grupos de almacenamiento dependen de dicho modelo.

Lo primero que debemos analizar es qué ofrece un único servidor y qué ofrece un grupo de almacenamiento. De forma somera, un servidor consta de ficheros, directorios, que almacenan bloques de datos y de metainformación, que permite gestionar estos bloques de datos para que sean accesibles a través del uso de los anteriores conceptos lógicos. Además, tiene un conjunto de operaciones que se utilizan como interfaz de acceso a los datos. En definitiva, un servidor consta de un conjunto de objetos, los cuales ofrecen una serie de métodos.

Análogamente, los grupos de almacenamiento deben ofrecer la misma imagen a las aplicaciones y de este modo, mantenerse transparentes a las mismas. Por tanto, los grupos de almacenamiento

ofrecen los mismos conceptos lógicos que un servidor, aunque ofrecen las ventajas analizadas en la sección 7.2.

Esto da respuesta a nuestra pregunta: el modelo de distribución de los datos condiciona los grupos de almacenamiento, porque éstos deben ofrecer una imagen única de la información. Es decir, el uso de grupos de almacenamiento debe ofrecer al sistema todas las propiedades que ofrece un servidor de forma aislada. Además, debe permitir ampliar sus funcionalidades, debido a que se define un nuevo nivel en la jerarquía de E/S. Por tanto, será necesario resolver todos los problemas planteados por la distribución de los datos de forma interna, quedando esta problemática transparente al usuario. De hecho, las aplicaciones de usuario sólo verán el grupo de almacenamiento y el nombre del fichero, encargándose el formalismo de grupos de almacenamiento de resolver la proyección de estos parámetros de entrada sobre la estructura física correspondiente.

Para definir el modelo de distribución de datos, vamos a plantear una serie de escenarios posibles de funcionamiento del sistema, que permitan establecer el alcance del mismo:

1. Creación de un grupo de almacenamiento a partir de servidores vacíos: En este caso, la formación del grupo es previa a la creación de ficheros o directorios en los servidores. Por tanto, el grupo ya está configurado antes de la distribución de la información.

2. Creación de un grupo de almacenamiento a partir de servidores no vacíos: El grupo de almacenamiento se compone de servidores que ya tienen ficheros. Redistribuir esta información entre los componentes del nuevo grupo es una opción muy costosa, por lo que será necesario tratar con ficheros distribuidos entre diferente número de servidores dentro de un mismo grupo de almacenamiento.

3. Unión de un servidor a un grupo de almacenamiento existente: Este caso constituye una generalización del anterior, ya que partimos de un servidor que se agrega a un grupo ya constituido y que, por tanto, tiene ya realizada la distribución de los ficheros entre los servidores. Como en el caso anterior, la redistribución no es una opción válida, ya que es muy ineficiente.

4. Creación de un grupo de almacenamiento a partir de dos grupos existentes: Este caso vuelve a constituir de nuevo una generalización del caso anterior. En este caso son diferentes grupos con diferentes distribuciones los que se unen para formar un nuevo grupo y de nuevo, se puede descartar la redistribución.

5. Eliminación de un servidor de un grupo de almacenamiento: En este caso, debido a que no se quiere perder la información almacenada en el servidor, es necesario redistribuir la información entre el resto de los componentes del grupo de almacenamiento. Esta operación no debe ser realizada muy frecuentemente, debido a que es muy costosa. Además es recomendable llevarla a cabo en horas de baja ocupación del sistema.

6. Eliminación de un grupo de almacenamiento: La información del grupo de almacenamiento debe ser redistribuida en otro grupo de almacenamiento diferente. Del mismo modo que el caso anterior, esta operación debe realizarse en horas de baja ocupación del sistema.

A continuación se va a proceder a analizar cada uno de estas situaciones, intentando dar una solución a los problemas que se plantean en cada una de ellas.

7.4.1. Creación de un grupo de almacenamiento a partir de servidores vacíos

Esta situación implica que la fase de configuración de los grupos de almacenamiento se lleva a cabo antes de la creación de ficheros y, por tanto, de la distribución de la información a través de dichos ficheros. En este caso, la distribución de los datos sobre el nuevo grupo de almacenamiento se realiza a través un conjunto de nodos inicialmente vacío y, por tanto, todos los ficheros de este grupo de almacenamiento se distribuyen de forma homogénea entre dichos nodos.

A continuación, se describe cuál es el funcionamiento interno de dicha distribución de la información.

7.4.2. Funcionamiento interno de la distribución de los datos

Para el funcionamiento de los grupos de almacenamiento, es necesario decidir dos aspectos relacionados con la distribución de la información y metainformación, a saber:

- Dónde se encuentra situada la metainformación, es decir, el *map-node* de un determinado fichero.
- Cuál es el nodo inicial a partir del cual se realiza el reparto de los datos de los diferentes ficheros.

Para tomar estas decisiones, el criterio que se va a tener en cuenta a la hora de decidir los anteriores parámetros es mantener el equilibrio de ocupación de almacenamiento. Debido a que los distintos discos que forman parte del sistema de almacenamiento pueden tener distintos tamaños, es necesario llevar un registro del espacio libre.

Las decisiones de diseño que se han tomado para resolver la problemática descrita son:

- Por motivos de simplicidad a la hora de acceder a la metainformación de los distintos ficheros, ésta se almacenará en un único nodo, que vamos a denominar **nodo maestro** y que será diferente para cada uno de los ficheros del grupo de almacenamiento, pudiendo ejercer este rol cualquiera de los nodos del grupo. Para asignar dicho papel, se utilizará un algoritmo, que debe cumplir dos características:

  1. Permitir una distribución homogénea de los nodos maestros.
  2. Dicho algoritmo debe ser rápido en ejecución, ya que se va a ejecutar cada vez que se requiera acceder a la metainformación.

  La descripción funcional del mismo es la siguiente:

```
ENTRADA: Nombre del fichero que se va a utilizar.
SALIDA:  Nodo maestro donde se encuentra almacenada la información
         del fichero de entrada.
PROCESO: masterNode(file)
```

  Esta función se ejecuta de forma interna en cada grupo de almacenamiento.

  masterNode() consiste en una función *hash* que debe utilizar el nombre del fichero como clave para el cálculo del nodo maestro. El carácter modular de la implementación de MAPFS_FS permite reemplazar dicha función por cualquier otra que respete los requisitos de diseño.

  El nombre del fichero en un sistema es único. Por tanto, en ese sentido no hay ningún problema en utilizarlo como información de entrada al algoritmo. El problema aparece cuando se renombra un fichero, ya que en este caso, el nodo maestro puede modificarse. En este caso, el problema

se puede resolver incrementando la funcionalidad de la función de renombrado, `mapRename()`. Para ello, esta función debe realizar las siguientes tareas:

1. Renombrar el fichero existente con el nuevo nombre.
2. Calcular el nodo maestro a partir del nuevo nombre.
3. Redistribuir la metainformación desde el antiguo nodo maestro al nuevo nodo maestro, en caso de que fuera diferente. Esta redistribución no es muy costosa, ya que la metainformación supone un porcentaje muy pequeño de los datos.

- Por el mismo motivo, el nodo inicial debe variar de forma homogénea, aunque en este caso se debe decidir dicho nodo teniendo en cuenta el espacio libre de los distintos nodos que forman el sistema. La descripción funcional del algoritmo que permite determinar el nodo inicial correspondiente a un fichero es la siguiente[3]:

```
ENTRADA: Nombre del fichero que se va a utilizar.
         Lista de capacidades de almacenamiento de los diferentes nodos.
SALIDA:  Nodo inicial donde se empiezan a almacenar los bloques de datos
         del fichero de entrada.
PROCESO: beginningNode(file,storageCapList)
```

El proceso puede llevarse a cabo a través del siguiente algoritmo:

1. Se recorre la lista de almacenamiento *storageCapList*[4] en busca del nodo con mayor capacidad de almacenamiento sobrante.
2. En el nodo resultante se comenzarán a almacenar los datos del fichero, en rodajas del tamaño especificado como configuración.
3. Se calcula el nodo maestro del fichero (*masterNode(file)*).
4. El identificador del nodo inicial se almacena en la metainformación que contiene el nodo maestro correspondiente.

Por tanto, es necesario definir una nueva entrada para la estructura que describe la metainformación, es decir, el *map-node*. De esta forma, la estructura queda representada en la figura 7.1.

7.4.3. Creación de un grupo de almacenamiento a partir de servidores no vacíos

Esta sección trata de la creación de un grupo de almacenamiento a partir de servidores que ya contienen ficheros, sean éstos distribuidos o no. La diferencia frente al caso anterior radica en que es necesario llevar a cabo un proceso de **reconstrucción** de la información si se desea homogeneizar los servidores que forman parte del grupo de almacenamiento, es decir, que los ficheros se distribuyan a través de todos los servidores pertenecientes a dicho grupo de almacenamiento. El problema de la reconstrucción de la información se trata en la siguiente sección ya que afecta a todos aquellos casos en los que la topología se ve modificada por cambios en alguno de los nodos.

[3] Nuevamente este algoritmo puede ser sustituido por otro diferente que cumpla las condiciones impuestas

[4] La lista de capacidades de almacenamiento de los diferentes nodos se puede calcular en cada instante a partir de la invocación a la llamada al sistema *s*tatfs(). Esta llamada tiene la cabecera *i*nt statfs(const char *path, struct statfs *buf) y devuelve información sobre un sistema de ficheros montado

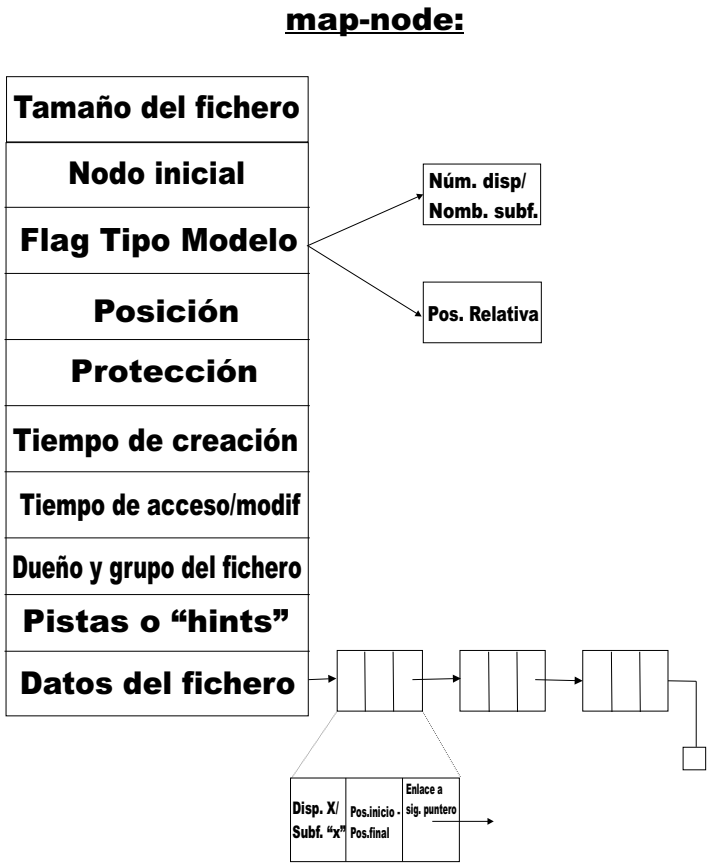


Figura 7.1: Estructura del *m*ap-node, incluyendo el nodo inicial

7.4.4. Reconstrucción de la información

Uno de los aspectos relacionados con la distribución de la información y metainformación que queda por decidir es la forma en la que se lleva a cabo la reconstrucción de la información en el caso de que algún nodo se añada a la topología inicial o que servidores que contienen ficheros formen un nuevo grupo de almacenamiento.

Una primera alternativa consiste en realizar la reconstrucción de la información mediante una técnica de *fuerza bruta*. Esta técnica consiste en la consecución de los siguientes pasos:

1. Realizar la lectura de todos los ficheros.
2. Escribir los datos leídos en nuevos ficheros teniendo en cuenta la nueva topología.
3. Borrar los ficheros antiguos.

Esta opción es muy poco eficiente. Teniendo en cuenta que en entornos de altas prestaciones es factible la posibilidad de incorporar nodos de forma puntual para ofrecer más servicios, el cambio de topología de una red puede ser una operación bastante utilizada. Esto unido a las facilidades que ofrecen los clusters para realizar toda la reconfiguración de forma dinámica [Buy99a], indica que la técnica de fuerza bruta no es una solución aceptable.

Por otro lado, al modificar la topología, las operaciones de E/S necesitan tener conocimiento de los nuevos cambios. La ventaja de redistribuir la información es que estos cambios son transparentes

para dichas operaciones.

Los parámetros que pueden verse modificados al cambiar la topología de la red son:

- El nodo maestro de un fichero. Eso es debido a que el número de nodos se ha modificado.
- El nodo inicial de un fichero. Eso es debido a que al añadir o eliminar nodos, las capacidades de almacenamiento del conjunto de nodos global se ven modificadas.

Los grupos de almacenamiento permiten llegar a una solución intermedia, de forma que sólo se lleve a cabo la redistribución de los datos si se elimina un nodo de un grupo de almacenamiento existente. Para evitar la redistribución, cada vez que se incluya un nodo en la topología, se creará un nuevo grupo de almacenamiento, que incluirá a los nodos existentes más el nuevo nodo. Las operaciones de E/S se realizarán sobre un determinado grupo de almacenamiento.

Por el contrario, si un nodo desaparece de la topología, se debe redistribuir su información para que se mantenga operativa. No obstante, esta reconstrucción se lleva sólo a nivel de su grupo de almacenamiento.

Por tanto, la ventaja que ofrecen los grupos de almacenamiento respecto a la reconstrucción y que puede añadirse a las ya descritas en la sección 7.2 es la simplicación del proceso de reconstrucción por los dos motivos ya mencionados, a saber:

1. Las operaciones de reconstrucción se restringen a nivel de grupo.
2. Además, en principio, la reconstrucción sólo se lleva a cabo en el caso de eliminación de nodos.

No obstante, evitar el proceso de reconstrucción en las operaciones de adición tiene un efecto colateral: si la topología cambia frecuentemente, se crea un gran número de grupos de almacenamiento con ligeras diferencias entre ellos. Para eliminar esta desventaja, el sistema proporciona una función de defragmentación de grupos de almacenamiento, cuya finalidad es la misma que la de la defragmentación de disco [WC00], [NST99], salvo que en este caso, la función permite redistribuir toda la información y homogeneizar los grupos de almacenamiento, evitando la fragmentación de los datos entre muchos grupos. Esta operación es proporcionada por MAPFS y se denomina `mapGroupDefragm()`. Su funcionamiento consiste en la implementación de la técnica de fuerza bruta que hemos descrito previamente. Esta operación de mantenimiento debe realizarse en horas de baja ocupación del sistema, para que no afecte a las operaciones de E/S.

Por tanto, la introducción de los grupos de almacenamiento modifica ligeramente las funciones que permiten calcular los nodos maestro e inicial. Para llevar a cabo dichas operaciones, se debe almacenar la información sobre la constitución de los grupos de almacenamiento. Las modificaciones son las siguientes:

- La función de obtención del nodo maestro:

```
ENTRADA: Nombre del fichero que se va a utilizar.
         Identificación del grupo de almacenamiento.
SALIDA:  Nodo maestro donde se encuentra almacenada la información
         del fichero de entrada.
PROCESO: masterNode(file, group)
```

  La implementación de un posible algoritmo es equivalente al anterior, pero teniendo en cuenta que se aplica dentro de un grupo de almacenamiento.

- Por otro lado, la función de obtención del nodo inicial puede describirse de la siguiente forma:

```
ENTRADA: Nombre del fichero que se va a utilizar.
         Lista de capacidades de almacenamiento de los diferentes nodos.
         Identificación del grupo de almacenamiento.
SALIDA:  Nodo inicial donde se empiezan a almacenar los bloques de datos
         del fichero de entrada.
PROCESO: beginningNode(file,storageCapList,group)
```

El proceso puede llevarse a cabo a través del siguiente algoritmo:

1. Se recorre la lista de almacenamiento *storageCapList* en busca del nodo con mayor capacidad de almacenamiento sobrante dentro del grupo de almacenamiento correspondiente.
2. En el nodo resultante se comenzarán a almacenar los datos del fichero, en rodajas del tamaño especificado como configuración.
3. Se calcula el nodo maestro del fichero (*masterNode(file,group)*) dentro del grupo de almacenamiento correspondiente.
4. El resultado de calcular el nodo inicial se almacena en la metainformación que contiene el nodo maestro correspondiente.

Esta es solamente una de las políticas utilizadas. El sistema MAPFS permite sustituir esta política por otra diferente.

Por tanto, de cara a constituir un grupo de almacenamiento a partir de servidores no vacíos y evitar la reconstrucción de la información, no debemos crear un único grupo, sino tantos grupos como diferentes distribuciones de los ficheros haya más uno que incluya a todos los servidores y que denominaremos **grupo principal**.

Definición 8 *Denominamos distribución de un fichero a la lista de servidores* $(S_1, S_2, \ldots, S_n)$ *a través de los cuales está homogéneamente repartido o distribuido.*

Definición 9 *Denominamos grupo principal correspondiente a un conjunto de servidores a aquél grupo de almacenamiento que incluye a todos los servidores de dicho conjunto y que tiene una misma distribución para todos los ficheros que alberga.*

Es muy probable que al constituir un grupo principal por primera vez, éste sea vacío. En otro caso, el grupo principal coincidiría con una distribución existente. En esta situación, el grupo principal sólo se computa una vez en el cálculo del número de grupos de almacenamiento, o bien como grupo principal, o bien como distribución existente.

En el caso más típico, si se unen n servidores que no tienen ningún fichero en común, será necesario crear $n+1$ grupos de almacenamiento, uno por cada servidor y otro que es el grupo principal, inicialmente vacío.

Como podemos ver, la introducción del concepto de grupo de almacenamiento diluye un poco el concepto de servidor, de forma que ahora la propiedad de los ficheros no es tanto del servidor, como del grupo de almacenamiento que contiene al mismo. Los servidores sirven como contenedores de la información y los grupos de almacenamiento como gestores de dicha información.

El grupo principal es el grupo visible por las aplicaciones, ya que el resto de los grupos se utilizan exclusivamente para evitar la degradación del sistema, eliminando la necesidad de reconstrucción de la información. Por esta razón, al resto de los grupos se les denomina **grupos invisibles**. También se les puede llamar **grupos secundarios**, por contraposición a grupo principal.

Definición 10 *Denominamos grupo invisible o grupo secundario a cualquier grupo de almacenamiento que no sea grupo principal de un conjunto de servidores.*

Observación 1 *El grupo principal correspondiente a un conjunto de servidores es un superconjunto de todos los grupos de almacenamiento que incluyan a alguno de esos servidores.*

7.4.5. Unión de un servidor a un grupo de almacenamiento existente

Un escenario frecuente en el uso de un cluster de nodos es añadir un nuevo nodo o servidor. En esta situación, dicho servidor puede agregarse a un grupo de almacenamiento ya existente. Se pueden dar dos casos:

1. El servidor está vacío. Es decir, no contiene ningún fichero.
2. El servidor no está vacío y, por tanto, es necesario seguir proporcionando acceso a los ficheros que contenga.

En el primer caso, y según vimos en la sección anterior, para evitar la reconstrucción de la información, tendremos que crear un único nuevo grupo de almacenamiento, el grupo principal, que contiene a todos los servidores. Se mantiene el grupo de almacenamiento original para la gestión de los ficheros de los que constara. En este caso, el grupo principal estará vacío.

En el segundo caso, debemos crear dos nuevos grupos de almacenamiento: un grupo de almacenamiento para albergar los ficheros del nuevo servidor y el grupo principal, que estará inicialmente vacío. Además, se ha de mantener el grupo original, como en el caso anterior.

7.4.6. Creación de un grupo de almacenamiento a partir de dos grupos existentes

Sean dos grupos de almacenamiento principales G_x y G_y a partir de los cuales deseamos crear un tercer grupo de almacenamiento que denominaremos **grupo unión** G_z. A fin de evitar la reconstrucción de los ficheros de ambos grupos de almacenamiento en uno nuevo, el grupo unión G_z estará constituido por todos los servidores que forman parte de los grupos originales G_x y G_y, manteniéndose los ficheros existentes hasta ese momento en dichos grupos. A partir de entonces, cualquier operación realizada sobre el nuevo grupo se realizará sobre todos los servidores que pertenezcan al mismo, salvo que se refieran a ficheros existentes en los grupos originales. Este comportamiento queda transparente a las aplicaciones de usuario que utilizan el sistema, ya que para dichas aplicaciones sólo existe un único grupo de almacenamiento G_z a partir de haberse llevado a cabo la unión, ya que es el nuevo grupo principal.

Obsérvese que la unión de grupos de almacenamiento se hace sobre grupos principales, ya que éstos son los únicos que ven las aplicaciones. No obstante, los grupos invisibles asociados a los grupos principales no desaparecen al realizar la unión de dos grupos de almacenamiento, sino que pasan a formar parte del nuevo grupo de almacenamiento como grupos invisibles. A su vez, los grupos principales G_x y G_y pasan también a ser grupos invisibles del nuevo grupo principal G_z. Es decir:

$$
\begin{aligned}
&G_x \cup G_y = G_z \\
&G_z \text{ es grupo principal} \\
&G_x \text{ y } G_y \text{ se convierten en grupos invisibles de } G_z \\
&\text{Si } G_w \text{ es grupo invisible de } G_x \text{ o } G_y \\
&\Longrightarrow G_w \text{ es grupo invisible de } G_z
\end{aligned}
$$

La única excepción a esta regla se da cuando alguno de los grupos originales G_x o G_y está vacío. En ese caso, el grupo vacío desaparece al hacer la unión.

Proposición 3 *Cada servidor del sistema pertenece a un único grupo principal.*

Demostración 3 *Por reducción al absurdo, supongamos que esta afirmación no es cierta. En ese caso, existirá al menos un servidor S_i que pertenezca a dos grupos principales diferentes G_x y G_y. De la definición de grupo principal se deduce que G_x y G_y son superconjuntos de todos los grupos de almacenamiento que incluyen a dichos servidores, incluidos ellos mismos. Por tanto, G_x es superconjunto de G_y y G_y es superconjunto de G_x, de lo que se deduce que G_x es igual a G_y. Esto contradice que los dos grupos principales sean diferentes. De este modo queda demostrado que cada servidor del sistema pertenece a un único grupo principal.*

Definición 11 *Se denomina* **relación de agrupación principal** *a la relación de pertenencia a un grupo principal. Se denota como R_{GP}. Esta relación se define del siguiente modo:*

$$S_i R_{GP} S_j \Longleftrightarrow S_i \in G_x \wedge S_j \in G_x$$
$$\text{siendo } G_x \text{ un grupo principal.}$$

Proposición 4 *Los grupos principales de un sistema constituyen una partición de los servidores existentes en el sistema.*

Demostración 4 *Para ello debemos demostrar las propiedades reflexiva, simétrica y transitiva de la relación de agrupación principal. La relación R_{GP} está basada en la relación de pertenencia, que es una relación de equivalencia. Por tanto, R_{GP} también lo es y los grupos principales constituyen una partición de los servidores del sistema, c.q.d.*

Observación 2 *Los grupos principales de un sistema son grupos ortogonales.*

Obsérvese que esta propiedad es muy interesante, ya que permite beneficiarse de las ventajas de ver los grupos principales como una partición de los servidores, lo que simplifica la gestión de los mismos.

A la aplicación que permite construir la asociación entre servidores y grupos de almacenamiento principales la denominaremos **función de agrupación principal**. Se denota como $\lambda_{GP}(S_i)$. Se cumple que:

$$\lambda_{GP}(S_i) = G_j \Longleftrightarrow S_i \in G_j$$
$$\wedge\, G_j \text{ es un grupo principal.}$$

Proposición 5 *La función de agrupación principal es una función inyectiva.*

Demostración 5 *Por reducción al absurdo, supongamos que esta afirmación no es cierta. En ese caso, existirá al menos un servidor S_i, tal que $\lambda_{GP}(S_i)$ es igual a dos grupos de almacenamiento G_x y G_y diferentes. Pero eso contradice a la proposición 3. Por tanto, la función de agrupación principal es una función inyectiva.*

Definición 12 *La matriz de una relación de agrupación principal se denomina* **matriz de agrupación principal** *y se construye a partir de la siguiente expresión:*

$$M_{GP} = (mgp_{ij}) donde\ mgp_{ij} = \begin{cases} 1 & si\ (S_i, S_j) \in R_{GP} \\ 0 & si\ (S_i, S_j) \notin R_{GP} \end{cases} \tag{7.5}$$

Al igual que cualquier matriz de agrupación correspondiente a una función de agrupación inyectiva, sólo es necesario almacenar la mitad de los elementos de esta matriz.

Por otro lado, tal y como se ha establecido en la creación de los grupos secundarios, se puede observar que existe una relación de orden entre los grupos secundarios del sistema y su correspondiente grupo principal. Esta relación de orden es una relación de "pertenencia", considerando los servidores que forman parte del grupo de almacenamiento. A esta relación la denominaremos **pertenencia grupal**. Más aún, cada grupo principal en conjunción con los grupos secundarios que contiene forma un **retículo**.

Definición 13 *Sea P_G una relación definida sobre los grupos de almacenamiento del sistema, que denominamos relación de pertenencia grupal. La relación P_G entre dos grupos de almacenamiento G_i y G_j se define como:*

$$G_i P_G G_j \Longleftrightarrow G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \subseteq G_j = \{S_{j(1)}, \ldots, S_{j(r)}\}$$

Proposición 6 *La relación de pertenencia grupal P_G es una relación de orden.*

Demostración 6 *Para demostrar que P_G es una relación de orden, tendremos que demostrar que cumple las propiedades reflexiva, antisimétrica y transitiva.*

- **propiedad reflexiva:**

$$\begin{aligned} &P_G \text{ es reflexiva sii } G_i P_G G_i \ \forall G_i \in \mathbf{G} \\ &\Longleftrightarrow G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \subseteq G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \end{aligned}$$

  Por tanto, cumple la propiedad reflexiva, ya que un conjunto siempre se incluye a sí mismo.

- **propiedad antisimétrica:** *Supongamos que $G_i P_G G_j$ y que $G_j P_G G_i$. En ese caso:*

$$\begin{aligned} &G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \subset G_j = \{S_{j(1)}, \ldots, S_{j(r)}\} \wedge \\ &G_j = \{S_{j(1)}, \ldots, S_{j(r)}\} \subset G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \\ &\Longrightarrow G_i = G_j \end{aligned}$$

  Por tanto, se cumple la propiedad antisimétrica.

- **propiedad transitiva:** *Supongamos que $G_i P_G G_j$ y que $G_j P_G G_t$. En ese caso:*

$$\begin{aligned} &G_i = \{S_{i(1)}, \ldots, S_{i(m)}\} \subset G_j = \{S_{j(1)}, \ldots, S_{j(r)}\} \wedge \\ &G_j = \{S_{j(1)}, \ldots, S_{j(r)}\} \subset G_t = \{S_{t(1)}, \ldots, S_{t(l))}\} \\ &\Longrightarrow G_i \subset G_t \end{aligned}$$

  debido a que la inclusión de conjuntos tiene la propiedad transitiva.

Por tanto, la relación de pertenencia grupal es una relación de orden, c.q.d.

Definición 14 *Se denomina grupo de almacenamiento nulo al grupo de almacenamiento que no contiene ningún servidor y se denota como* $\emptyset$*. Por definición, cualquier grupo incluye al grupo nulo.*

Definición 15 *Se denomina retículo principal de un grupo principal al retículo formado por el grupo principal, sus grupos secundarios y el grupo nulo y que tiene como relación de orden la relación de pertenencia grupal* P_G*. Al conjunto formado por el grupo principal* GP_i*, sus grupos secundarios y el grupo nulo se le denota como* $\sum GP_i$ *y al retículo principal* $(\sum GP_i, P_G)$*.*

Proposición 7 *Un retículo principal es un retículo.*

Demostración 7 *Para demostrarlo, debemos probar que:*

- $(\sum GP_i, P_G)$ *es un conjunto ordenado: Esta afirmación ya ha sido demostrada al probar que la relación de pertenencia grupal es una relación de orden.*
- $\forall G_x, G_y \in \sum GP_i \ \exists \ sup\{G_x, G_y\}$.
- $\forall G_x, G_y \in \sum GP_i \ \exists \ inf\{G_x, G_y\}$.

Para demostrar las dos últimas afirmaciones, podemos encontrarnos en dos situaciones diferentes:

1. G_x *y* G_y *son comparables: Eso implica que* $G_x P_G G_y$ *o* $G_y P_G G_x$*. Por tanto, en este caso, el supremo es el mayor de los grupos de almacenamiento y el ínfimo el menor según el orden establecido por la relación de pertenencia grupal.*

2. G_x *y* G_y *no son comparables: En este caso, debemos demostrar que existe al menos un conjunto* G_z*, tal que* $G_x P_G G_z$ *y* $G_y P_G G_z$*. Si demostramos esto, tendremos la seguridad de que existe un conjunto que contiene a ambos y por tanto, quedara demostrado que existe un máximo. Por tanto, la existencia de supremo estaría asegurada. El grupo principal* GP_i *cumple siempre esto, ya que como vimos en la observación 1, el grupo principal es un superconjunto de todos sus grupos secundarios. Por tanto, queda demostrada la existencia de supremo. Por otro lado, debemos demostrar que existe al menos un conjunto* G_z*, tal que* $G_z P_G G_x$ *y* $G_z P_G G_y$*. Si demostramos esto, tendremos la seguridad de que existe un conjunto contenido por ambos y por tanto, que existe un mínimo. De este modo, la existencia de ínfimo estaría asegurada. El grupo que siempre cumple esto es el grupo nulo, ya que por definición, cualquier grupo incluye al grupo nulo. Luego, queda demostrada la existencia de ínfimo.*

Un retículo principal es un retículo, c.q.d.

Definición 16 *Un retículo principal se puede definir también como una terna* $(\sum GP_i, \vee, \wedge)$ *donde las operaciones disyuncion* $(\vee)$ *y conjunción* $(\wedge)$ *consisten en:*

$$\forall S_x, S_y \in \sum GP_i, \ S_x \vee S_y = sup\{S_x, S_y\}, \ S_x \wedge S_y = inf\{S_x, S_y\}$$

En el caso de los retículos principales, las operaciones de disyunción y conjunción coinciden con la unión de conjuntos $(\cup)$ *e intersección de conjuntos* $(\cap)$ *respectivamente.*

Observación 3 *Cada partición del sistema forma una estructura de retículo sobre el conjunto de servidores de dicha partición. Al conjunto de todos los retículos le denominamos* **partición reticular**.

Ejemplo 1

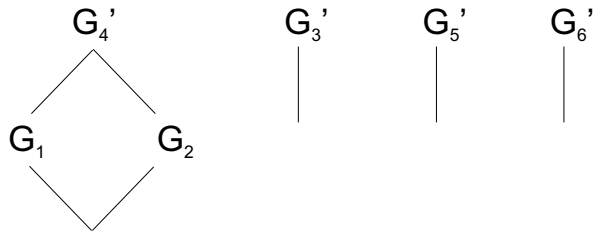


Figura 7.2: Partición reticular correspondiente al escenario 1

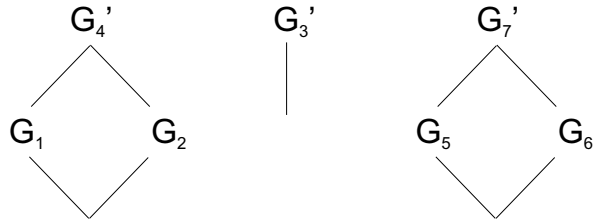


Figura 7.3: Partición reticular correspondiente al escenario 2

- *Escenario 1*[5]*:*

$$G_1 = \{S_1, S_2\}$$
$$G_2 = \{S_3, S_4\}$$
$$G'_3 = \{S_5, S_6\}$$
$$G'_4 = \{S_1, S_2, S_3, S_4\}$$
$$G'_5 = \{S_7, S_8\}$$
$$G'_6 = \{S_9, S_{10}\}$$

La figura 7.2 muestra la partición reticular del escenario 1.

- *Escenario 2: Se unen los grupos* G_5 *y* G_6*. Se añade un servidor al grupo* G_3*.*

$$G_1 = \{S_1, S_2\}$$
$$G_2 = \{S_3, S_4\}$$
$$G'_3 = \{S_5, S_6, S_{11}\}$$
$$G'_4 = \{S_1, S_2, S_3, S_4\}$$
$$G_5 = \{S_7, S_8\}$$
$$G_6 = \{S_9, S_{10}\}$$
$$G'_7 = \{S_7, S_8, S_9, S_{10}\}$$

La figura 7.3 muestra la partición reticular del escenario 2.

[5]Los grupos que contienen una ' son los grupos principales

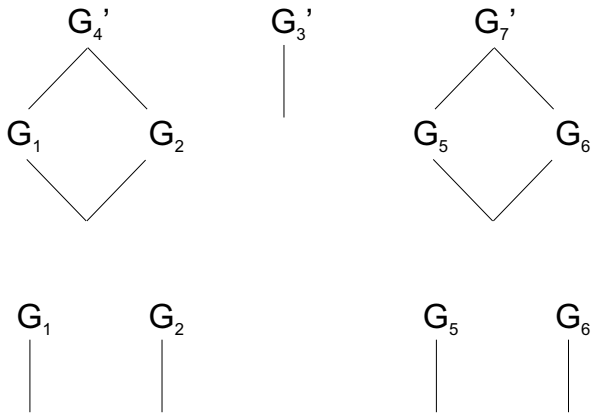


Figura 7.4: Subretículos correspondientes a los grupos secundarios

Proposición 8 *Cada grupo secundario constituye el máximo de algún subretículo del retículo principal y, por tanto, cada retículo principal tiene al menos tantos subretículos como grupos secundarios.*

Demostración 8 *Dado un retículo principal* $(\sum GP_i, P_G)$*, por cada grupo secundario* $GS_{i(j)}$ *se puede construir un subretículo que tiene como únicos elementos dicho grupo secundario y el grupo nulo. Denotaremos este subconjunto como* $GSR_{i(j)}$*. Se puede comprobar que este subconjunto constituye un subretículo:*

1. *El subconjunto es retículo, porque está ordenado (el grupo nulo está relacionado con cualquier grupo secundario) y además tiene supremo (el grupo secundario) e ínfimo (el grupo nulo).*

2. *Además se ha de comprobar que:*

$$\forall x, y \in GSR_{i(j)},\ sup_{\sum GP_i}\{x, y\} = sup_{GSR_{i(j)}}\{x, y\}$$
$$\wedge\ inf_{\sum GP_i}\{x, y\} = inf_{GSR_{i(j)}}\{x, y\}$$

En este caso, x *e* y *sólo pueden corresponder con el grupo secundario y con el grupo nulo y se cumple la proposición anterior, ya que:*

$$sup_{\sum GP_i}\{\emptyset, S_{i(j)}\} = S_{i(j)}$$
$$sup_{SR_i}\{\emptyset\}, S_{i(j)}\} = S_{i(j)}$$
$$inf_{\sum GP_i}\{\emptyset, S_{i(j)}\} = \emptyset$$
$$inf_{SR_i}\{\emptyset, S_{i(j)}\} = \emptyset$$

De esta forma, queda demostrado que al menos existen tantos subretículos como grupos secundarios.

Ejemplo 2 *En la figura 7.4 se muestran los subretículos que se pueden construir a partir de los grupos secundarios.*

7.4.7. Eliminación de un servidor de un grupo de almacenamiento

Las operaciones de eliminación son las únicas operaciones que exigen la redistribución de los ficheros, a fin de no perder la información almacenada en el componente eliminado. Como se ha mencionado

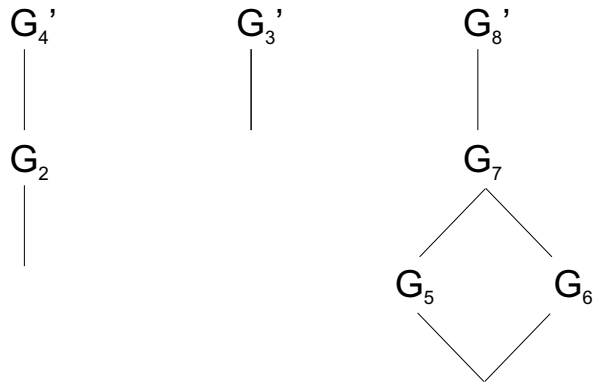


Figura 7.5: Partición reticular correspondiente al escenario 3

anteriormente, este tipo de operaciones no deben ser realizadas muy frecuentemente debido a que son muy costosas. Por este motivo, es recomendable llevarlas a cabo en horas de baja ocupación del sistema.

En el caso de eliminación de un servidor de un grupo de almacenamiento, los ficheros son redistribuidos entre el resto de los componentes del grupo de almacenamiento. La metainformación correspondiente debe ser regenerada, ya que al modificarse la topología, es necesario recalcular el nodo maestro y el nodo inicial. Además, el grupo de almacenamiento principal queda modificado al eliminarse el servidor.

Puede ocurrir que el servidor eliminado pertenezca a algún grupo de almacenamiento invisible asociado al principal. En ese caso, es necesario también redistribuir la información de tales grupos de almacenamiento en el nuevo grupo de almacenamiento principal y eliminar los primeros.

Ejemplo 3

- *Escenario 3: Se elimina el servidor S_1. Eso implica que el grupo secundario G_1 desaparece. Después se une el servidor S_1 (vacío) al grupo G_7.*

$$
\begin{aligned}
G_2 &= \{S_3, S_4\} \\
G'_3 &= \{S_5, S_6\} \\
G'_4 &= \{S_2, S_3, S_4\} \\
G_5 &= \{S_7, S_8\} \\
G_6 &= \{S_9, S_{10}\} \\
G_7 &= \{S_7, S_8, S_9, S_{10}\} \\
G'_8 &= \{S_1, S_7, S_8, S_9, S_{10}\}
\end{aligned}
$$

La figura 7.5 muestra la partición reticular del escenario 3.

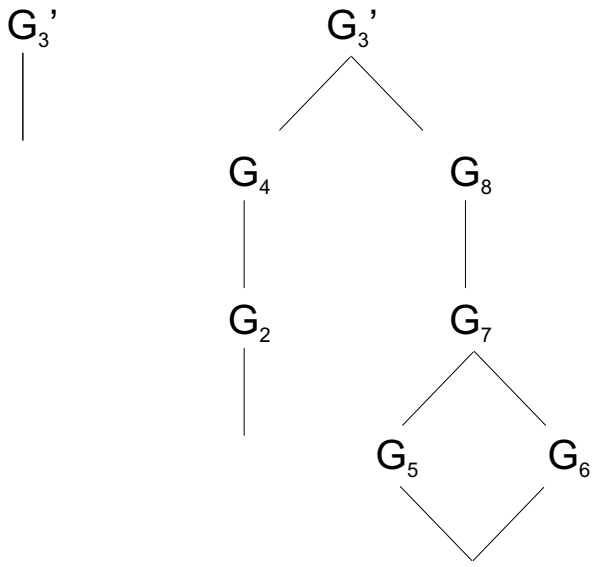


Figura 7.6: Partición reticular correspondiente al escenario 4

- *Escenario 4: Se unen los grupos G_4 y G_8.*

$$G_2 = \{S_3, S_4\}$$
$$G'_3 = \{S_5, S_6\}$$
$$G_4 = \{S_2, S_3, S_4\}$$
$$G_5 = \{S_7, S_8\}$$
$$G_6 = \{S_9, S_{10}\}$$
$$G_7 = \{S_7, S_8, S_9, S_{10}\}$$
$$G_8 = \{S_1, S_7, S_8, S_9, S_{10}\}$$
$$G'_9 = \{S_1, S_7, S_8, S_9, S_{10}, S_2, S_3, S_4\}$$

La figura 7.6 muestra la partición reticular del escenario 4.

7.4.8. Eliminación de un grupo de almacenamiento

Esta operación se realiza sobre grupos de almacenamiento principales. Aquellos grupos invisibles asociados al grupo de almacenamiento que se va a eliminar, también deben ser eliminados. Por tanto, tanto la información del grupo de almacenamiento principal como la de los grupos invisibles asociados debe ser redistribuida en un grupo de almacenamiento principal diferente, recalculando los nodos maestros e iniciales de cada fichero en dicho grupo principal. La selección del grupo de almacenamiento sobre el cual se va a realizar la redistribución se debe calcular en base a las capacidades de los grupos de almacenamiento existentes, de modo similar al cálculo del nodo inicial. La especificación de esta operación es la siguiente:

```
ENTRADA: Grupo de almacenamiento a extinguir.
         Lista de capacidades de almacenamiento de los diferentes grupos
```

```
         principales de almacenamiento.
SALIDA:  Grupo de almacenamiento donde se almacenarán los datos del
         grupo de almacenamiento a extinguir.
PROCESO: selectionNewStorageGroup(group,storageGroupCapList)
```

El proceso puede llevarse a cabo a través del siguiente algoritmo:

1. Se recorre la lista de almacenamiento *storageGroupCapList* en busca del grupo de almacenamiento con mayor capacidad de almacenamiento sobrante. Esta lista se calcula como se indica en la sección 7.6.

2. En el grupo resultante se comenzará a distribuir los ficheros almacenados en el grupo *group.*

7.5. Operaciones de grupos

Unido al concepto de grupo, existe un conjunto de operaciones que permiten gestionar dichas entidades. Las operaciones de grupos modifican las relaciones existentes entre los diferentes grupos de almacenamiento. Como ya hemos visto, la función de agrupación principal asociada es inyectiva y la relación de agrupación principal constituye una relación de equivalencia. Siempre nos va a interesar mantener esta propiedad, de cara a beneficiarnos de la representación de dicha relación.

Del mismo modo que los grupos de almacenamiento, también existen dos tipos de operaciones de grupos:

- **Operaciones ortogonales:** Son las operaciones que mantienen las propiedades que permiten que una determinada relación de agrupación R_G sea una relación de equivalencia y, por tanto, que el conjunto cociente R_G/S sea una partición del conjunto de servidores S.

- **Operaciones no ortogonales:** Son las operaciones que "rompen" alguna de las propiedades de una relación de equivalencia R_G y por tanto, convierten a un grupo de almacenamiento ortogonal en no ortogonal.

Estas operaciones permiten modificar las relaciones existentes entre servidores y grupos de almacenamiento[6]. Realmente son funciones que permiten modificar la función de agrupación, por lo que se trata de "metafunciones" que operan sobre el dominio de funciones: $O : \mathcal{F} \rightarrow \mathcal{F}$. Estas funciones son:

- `mapJoin(`S_x`,`G_y`)`: Permite que un servidor se una a un grupo de almacenamiento.

  Si denominamos $\lambda^{'}_{GP}(S_i)$ a la función de agrupación antes de realizar la operación `mapJoin()` y $\lambda_{GP}(S_i)$ a la función de agrupación después de realizar dicha operación, se cumple que:

$$\lambda^{'}_{GP}(S_x) = G_z \wedge \text{mapJoin}(S_x, G_y) \Longrightarrow \lambda_{GP}(S_x) = G_y$$

  Esta operación equivale a realizar un cambio de grupo de almacenamiento, operación que es llevada a cabo por la función `mapChange()` que veremos más adelante. La única diferencia es que la operación `mapJoin()` no necesita conocer el estado anterior, es decir, no necesita saber a qué grupo de almacenamiento estaba ligado previamente el servidor. De hecho, esta función también se puede utilizar para inicializar la relación de agrupación y, en este caso, el servidor no se encuentra inicialmente vinculado a ningún grupo de almacenamiento. En esta situación se dice que el servidor pertenece exclusivamente a su **grupo unario**.

[6]Nos referimos en este contexto a grupos de almacenamiento principales, ya que son los únicos grupos visibles

Definición 17 *Se denomina grupo unario de un servidor S_i al grupo de almacenamiento que incluye sólo a dicho servidor. Se denota como $\{S_i\}$.*

Por tanto, la semántica de la operación `mapJoin()` para inicialización es la siguiente:

$$\lambda_{GP}^{'}(S_x) = \{S_x\} \wedge \text{mapJoin}(S_x, G_y) \Longrightarrow \lambda_{GP}(S_x) = G_y$$

- `mapBreak(`S_x`,`G_y`)`: Desvincula a un servidor de un grupo de almacenamiento. El problema de aplicar esta operación es que si no se realiza posterior e inmediatamente una operación `mapJoin()`, que incluya al servidor en un grupo de almacenamiento, se eliminan las propiedades que hacen que R_G sea una relación de equivalencia y, por tanto, que el conjunto cociente R_G/S sea una partición del conjunto S, debido a que uno de los servidores queda desligado. Esto es debido a que la operación `mapBreak()` es una operación no ortogonal. Para evitar este problema, existen dos alternativas:

  1. Utilizar los grupos unarios en la definición de la operación. De este modo, desvincular un servidor de un grupo de almacenamiento equivale a eliminar el servidor de dicho grupo, construyendo su grupo unario y, por tanto, hacerle partícipe de un único grupo principal. Como se puede comprobar, los grupos unarios se utilizan para modificar el dominio de servidores **S** sobre el cual se define la relación de agrupación. En el caso de la operación `mapJoin()`, los grupos unarios permiten añadir un servidor a dicho dominio, mientras que en el caso de la operación `mapBreak()`, permiten eliminar de una forma gradual un servidor de dicho dominio.
  2. Modificar la relación entre un servidor y un grupo de almacenamiento, utilizando la operación `mapChange()`, que se describe a continuación. Esta es la opción más adecuada, ya que se trata de una operación ortogonal, que no modifica el dominio de servidores **S**.

- `mapChange(`S_x`,`G_y`,`G_z`)`: Modifica la función de agrupación, eliminando la relación que existe entre el servidor S_x y el grupo de almacenamiento G_y y relacionándolo con el grupo G_z. Esta operación equivale a hacer la siguiente secuencia de sentencias:

$$mapBreak(S_x, G_y);$$
$$mapJoin(S_x, G_z);$$

  La ventaja de agrupar las dos funciones en una sóla es que ésta última se puede definir como una **operación atómica** a nivel de sistema, es decir, definir que su ejecución se lleve a cabo de forma completa e indivisible en el sistema.

  Si de nuevo denominamos $\lambda_G^{'}(S_i)$ a la función de agrupación antes de realizar la operación `mapChange()` y $\lambda_G(S_i)$ a la función de agrupación después de realizar dicha operación, se cumple que:

$$\lambda_G^{'}(S_x) = G_y \wedge \text{mapChange}(S_x, G_y, G_z) \Longrightarrow \lambda_G(S_x) = G_z \tag{7.6}$$

Proposición 9 *La operación* `mapChange()` *es una operación ortogonal y, por tanto, no modifica las propiedades de la relación de agrupación R_G.*

Demostración 9 *Sea $R_G^{'}$ una relación de equivalencia que representa la relación de agrupación antes de aplicarle la operación* `mapChange()`*. Al ser una relación de equivalencia cumple las propiedades reflexiva, simétrica y transitiva.*
Sea R_G la relación de agrupación después de aplicarle la operación `mapChange()`*.*
Sean $\lambda_G^{'}$ y λ_G las funciones de agrupación correspondiente a la asociación existente entre servidores y grupos de almacenamiento antes y después de aplicar la operación `mapChange()`*.*

Veamos la demostración de que R_G conserva las propiedades correspondientes:

- **Propiedad reflexiva:**

  Partimos de que $R_G^{'}$ es reflexiva y que se ha realizado la operación `mapChange(`S_x`,`G_y`,`G_z`)`*:*

  $$\text{Según la ecuación (7.2) } R_G \text{ es reflexiva sii } \exists t / s \in G_t$$

  Se pueden dar dos casos:

  1. $s \neq S_x$*:*

     $$R_G^{'} \text{ es reflexiva } \Longrightarrow sR_G^{'}s.$$
     $$\text{La relación de agrupación no ha cambiado para el elemento } s$$
     $$\Longrightarrow sR_Gs \Longrightarrow R_G \text{ es reflexiva}$$

  2. $s = S_x$*:*

     $$mapChange(S_x, G_y, G_z) \Longrightarrow \lambda_G(S_x) = G_z$$
     $$\Longrightarrow \exists t = z / S_x \in G_t$$
     $$\Longrightarrow R_G \text{ es reflexiva}$$

  Luego la relación R_G es reflexiva c.q.d.

- **Propiedad simétrica:**

  De nuevo partimos de que $R_G^{'}$ es simétrica y que se ha realizado la operación mapChange(S_x,G_y,G_z).

  Se pueden dar cuatro casos:

  1. $s \neq S_x \wedge u \neq S_x$*:*

     $$R_G^{'} \text{ es simétrica sii } sR_G^{'}u \Longleftrightarrow uR_G^{'}s.$$
     $$\text{La relación de agrupación no ha cambiado para los elementos } s \text{ y } u$$
     $$\Longrightarrow (sR_Gu \Longrightarrow uR_Gs) \Longrightarrow R_G \text{ es reflexiva}$$

  2. $s = S_x \wedge u \neq S_x$*:*

     $$\lambda_G(s) = G_z. \text{ Si } sR_Gu \Longleftrightarrow \lambda_G(s) = \lambda_G(u) = G_z \Longleftrightarrow uR_Gs$$
     $$\Longrightarrow R_G \text{ es simétrica}$$

  3. $s \neq S_x \wedge u = S_x$*: Demostración equivalente al anterior caso.*

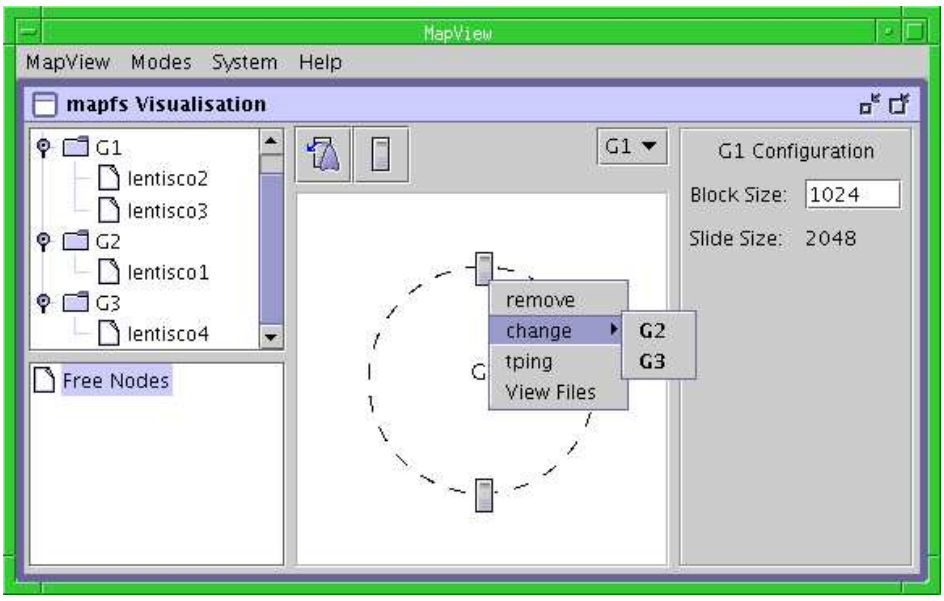


Figura 7.7: Interfaz gráfica del sistema MAPFS que muestra la topología del sistema

4. $s = S_x \wedge u = S_x$*:*

$$s = u \Longrightarrow sR_G u \wedge uR_G s \text{ por la propiedad reflexiva}$$

Luego la relación R_G *es simétrica c.q.d.*

- **Propiedad transitiva:** *Partimos de que* $R_G^{'}$ *es transitiva, la función de agrupación* $\lambda_G^{'}$ *es inyectiva y que se ha realizado la operación* $mapChange(S_x, G_y, G_z)$*.*

  Si consideramos que λ_G *es una aplicación inyectiva, la transitividad de* R_G *queda demostrada, a saber:*

$$\begin{aligned} sR_G u &\Longrightarrow \lambda_G(s) = G_x = \lambda_G(u) \\ uR_G v &\Longrightarrow \lambda_G(u) = G_x = \lambda_G(v) \\ &\Longleftrightarrow \lambda_G(s) = \lambda_G(v) \Longleftrightarrow sR_G v \end{aligned}$$

  Por tanto, lo único que debemos demostrar es que λ_G *es inyectiva.*
  Fijándonos en la semántica de la operación `mapChange()` *(ecuación (7.6)) y partiendo de que la función* $\lambda_G^{'}$ *es inyectiva, el único cambio que existe entre esta función y la nueva función de agrupación* λ_G *se lleva a cabo sobre el elemento* S_x*. Pero* λ_G *define un único valor para dicho elemento. De aquí deducimos que* λ_G *es una aplicación inyectiva.*

  Luego la relación R_G *es transitiva c.q.d.*

De esta forma, queda demostrado que la operación `mapChange()` *es una operación ortogonal.*

La figura 7.7 muestra la interfaz del sistema MAPFS. A través de dicha interfaz, se pueden realizar cambios sobre la topología del sistema.

Por otro lado, también se ha definido un conjunto de operaciones sobre grupos de almacenamiento, a saber:

- Operación *mapUnion(G_x, G_y)*, que permite calcular la unión de dos grupos de almacenamiento existentes, como queda descrito en la sección 7.4.5. Esta operación puede utilizarse usando como parámetros los índices de los grupos de almacenamiento entre los cuales se va a calcular la unión, es decir `mapUnion(i,j)` calcula la unión de los grupos de almacenamiento G_i, G_{i+1}, ..., G_j.
- Operación `mapIntersection`(G_x, G_y), que permite calcular el grupo de almacenamiento intersección de dos grupos de almacenamiento. De forma análoga a la operación anterior, `mapIntersection()` también puede utilizarse usando como parámetros los índices de los grupos de almacenamiento entre los cuales se va a calcular la intersección.
- Operación `mapComplementary`(G_x), que permite calcular el grupo de almacenamiento complementario de G_x, es decir el grupo G_y, que contiene todos los servidores existentes salvo los que contiene el grupo G_z.
- Operación `mapSubstract`(G_x, G_y), que devuelve un grupo de almacenamiento que contiene aquellos servidores existentes en el grupo G_x y no existentes en el grupo G_y.

Hay que tener cuidado al utilizar estas funciones, ya que en general no se trata de operaciones ortogonales, aunque utilizadas de una forma controlada, permiten mantener las propiedades de la relación de agrupación principal. Además, estas operaciones son útiles para definir distintos conjuntos y propiedades.

Proposición 10 *Sustituir dos grupos de almacenamiento principales por la unión de ambos (a través del uso de la operación* `mapUnion()`*) constituye una operación ortogonal y, por tanto, no modifica las propiedades de la relación de agrupación principal R_{GP}.*

Demostración 10 *Partimos de una relación de agrupación principal R'_{GP}, que es una relación de equivalencia. Sean G_x y G_y grupos de almacenamiento que van a ser sustituidos por el grupo unión de ambos G_z. Si denotamos R_{GP} a la relación de agrupación principal resultante de llevar a cabo esta operación, vamos a demostrar que esta relación también es de equivalencia y, por tanto, que la operación es ortogonal. Nos basta con probar que λ_{GP} es inyectiva.*

$$\lambda_{GP} \text{ es inyectiva} \iff \forall S_i \ \exists \text{ un único grupo } G_w / S_i \in G_w$$
$$\lambda_{GP}(S_i) = G_z, \text{ si } S_i \text{ pertenecía a } G_x \text{ o a } G_y$$
$$\lambda_{GP}(S_i) = \lambda'_{GP}(S_i), \text{ si } S_i \text{ pertenecía a } G_x \text{ o a } G_y$$

Para los dos casos existentes:

- *S_i pertenecía a G_x o a G_y:*

$$\lambda_{GP}(S_i) = G_z \wedge \lambda_{GP}(S_i) \neq G_w \forall w \neq z, \text{ ya que } \lambda'_{GP} \text{ es inyectiva}$$

- *S_i no pertenecía a G_x ni a G_y:*

$$\lambda_{GP}(S_i) = \lambda_{GP}(S_i) \neq G_z, \text{ porque ese grupo es nuevo y no estaba definido previamente}$$

Por tanto, λ_{GP} es inyectiva. De ahí se deduce que R_{GP} es una relación de equivalencia y la operación definida ortogonal c.q.d.

Proposición 11 *La intersección de dos grupos de almacenamiento principales es vacía*

Demostración 11 *Por reducción al absurdo, supongamos que no es cierto. Por tanto, existen dos grupos de almacenamiento principales que comparten al menos un servidor S_i. Pero eso contradice que la función de agrupación principal sea inyectiva. Por tanto, la intersección de dos grupos de almacenamiento principales es vacía, c.q.d.*

Proposición 12 *Dado un conjunto de n servidores S_i, una relación de agrupación principal R_{GP} que particiona el conjunto de servidores en m grupos de almacenamiento G_j, se cumple que $\forall j \in 1, \ldots, m$:*

$$G_j = \ mapComplementary\ (mapSubstract(mapUnion(1, m), G_j))$$

Demostración 12

$mapUnion(1, m) \equiv S$, siendo S el conjunto de servidores existentes, ya que los grupos principales forman una partición del conjunto de servidores S

$mapSubstract(S, G_j) \equiv$ Conjunto de grupos principales de almacenamiento, salvo G_j por definición de la operación mapSubstract()

$mapComplementary(mapSubstract(S, G_j)) \equiv G_j$ por definición de la operación mapComplementary()

Finalmente, debido a que los grupos de almacenamiento permiten la gestión, el almacenamiento y la recuperación de ficheros, la interfaz del sistema de ficheros queda modificada, para dar cabida al concepto de grupo de almacenamiento. Estas modificaciones se muestran en la próxima sección.

7.5.1. Modificación de la matriz de relación por parte de las operaciones de grupos

Las operaciones de grupos implican la modificación de la función de agrupación principal, lo que a su vez supone cambiar la relación de dicha agrupación y, por tanto, la matriz de agrupación principal que almacena el cliente MAPFS. Véamos cómo afectan las diferentes operaciones a esta matriz[7]:

- Operación `mapJoin()`: Las modificaciones sobre la matriz de agrupación dependen del objetivo con el cual se utilice esta función, a saber:
  - Si se utiliza para la inicialización de la relación de agrupación, el cambio que implica sobre la matriz es la introducción de unos en el lugar que corresponde a la intersección entre el servidor y los servidores del grupo de almacenamiento asociado y ceros en el resto. Es decir:

$$\lambda'_{GP}(S_x) = \emptyset \wedge \text{mapJoin}(S_x, G_y) \Longrightarrow mgp_{xw} = \begin{cases} 1 & \forall \text{w} \ / \ mgp_{xw} \in M_{GP} \wedge S_w \in G_y \\ 0 & \forall \text{u} \ / \ mgp_{xu} \in M_{GP} \wedge S_u \notin G_y \end{cases}$$

  - Si se utiliza para la modificación de la relación que existe entre un servidor y un grupo de almacenamiento, el cambio que implica sobre la matriz es la introducción de unos en el lugar que corresponde a la intersección entre el servidor y los servidores del nuevo grupo de almacenamiento y la inserción de ceros en el lugar que corresponde a la intersección entre el servidor y los servidores del antiguo grupo de almacenamiento al que estaba asociado; es decir:

[7] En este contexto, cuando se habla de grupo de almacenamiento, se refiere a grupo de almacenamiento principal

$$\lambda'_{GP}(S_x) = G_z \wedge \text{mapJoin}(S_x, G_y) \Longrightarrow mgp_{xw} = \begin{cases} 1 & \forall \text{w} \ / \ mgp_{xw} \in M_{GP} \wedge S_w \in G_y \\ 0 & \forall \text{u} \ / \ mgp_{xu} \in M_{GP} \wedge S_u \in G_z \end{cases} \tag{7.7}$$

- Operación `mapBreak()`: Esta operación implica la introducción de un 0 en la posición de la matriz correspondiente a la intersección entre el servidor y los servidores del correspondiente grupo de almacenamiento, es decir:

$$\lambda'_{GP}(S_x) = G_y \wedge \text{mapBreak}(S_x, G_y) \Longrightarrow mgp_{xw} = 0 \ \forall \text{w, donde } mgp_{xw} \in M_{GP} \wedge S_w \in G_y$$

- Operación `mapChange()`: La aplicación de esta operación equivale a la ecuación (7.7):

$$\lambda'_{GP}(S_x) = G_z \wedge \text{mapChange}(S_x, G_z, G_y) \Longrightarrow mgp_{xw} = \begin{cases} 1 & \forall \text{w} \ / \ mgp_{xw} \in M_{GP} \wedge S_w \in G_y \\ 0 & \forall \text{u} \ / \ mgp_{xu} \in M_{GP} \wedge S_u \in G_z \end{cases}$$

- Operación `mapUnion()`: La aplicación de esta operación implica que los servidores de los dos grupos de almacenamiento quedan relacionados, y, por tanto, la matriz de agrupación contendrá un uno en la intersección entre todos esos servidores.

$$\lambda'_{GP}(S_x) = G_w \wedge \lambda'_{GP}(S_y) = G_u \wedge \text{mapUnion}(G_w, G_u) \Longrightarrow mgp_{xy} = 1$$
$$\text{donde } mgp_{xy} \in M_{GP}$$

7.6. Parámetros de agrupación de MAPFS

Los grupos de almacenamiento permiten la gestión del almacenamiento y recuperación de información en el sistema de ficheros MAPFS.

Hay varios factores relacionados con los grupos de almacenamiento que se pueden ajustar de forma que se optimice el funcionamiento de dicho sistema. Estos factores permiten el análisis de determinadas características que pueden afectar al rendimiento de las aplicaciones y están basados en los grupos de almacenamiento principales, sin tener en cuenta los grupos de almacenamiento invisibles o secundarios. Esto no supone ninguna limitación, por dos motivos:

- Las aplicaciones sólo ven los grupos de almacenamiento principales.
- La operación `mapGroupDefragm()` permite la defragmentación de los grupos de almacenamiento, de forma que los grupos invisibles desaparecen, quedando sólo los grupos de almacenamiento principales. Como se mencionó previamente, a pesar de ser una operación costosa, puede ser realizada durante horas de baja ocupación del sistema.

Por este motivo, para calcular estos parámetros se utilizará la relación de agrupación principal, que por simplicidad a partir de este momento se va a denotar como relación de agrupación.

A continuación se describen los factores o parámetros de agrupación:

- **Factor de agrupación**. Se denota como $\alpha_G(\mathbf{S}, \mathbf{G})$, donde $\mathbf{S}$ es el conjunto de servidores y $\mathbf{G}$ el conjunto de grupos de almacenamiento. Consiste en el nivel de agrupamiento que tienen los

servidores. Para calcular este factor, se halla el ratio entre el número de servidores y el número de grupos de almacenamiento. Suponiendo que existen p servidores y n grupos, la expresión que permite calcular el factor de agrupación es:

$$\alpha_G(\mathbf{S},\mathbf{G}) = \frac{\sum_{i,j} S_{i(j)}}{\sum_{z=0}^{n} G_z} = p/n$$

En los casos límites, el valor del factor de agrupación oscila entre los siguientes valores:

- $\alpha_G(\mathbf{S},\mathbf{G}) = 1 \iff$ p=n, es decir, hay un grupo de almacenamiento por cada uno de los servidores. En este caso, el uso de los grupos de almacenamiento no aporta ninguna ventaja, ya que un grupo corresponde a un servidor y viceversa.
- $\alpha_G(\mathbf{S},\mathbf{G}) = p \iff$ n=1, es decir, sólo existe un grupo de almacenamiento. En este caso, todos los servidores son tratados de la misma forma desde el punto de vista del almacenamiento y se utilizan como copias exactas. Es lo que denominaremos **servidores clones**.

- **Nivel de homogeneidad de agrupación**. Se denota como $\beta_G(\mathbf{S},\mathbf{G})$ y consiste en medir cómo se distribuyen los servidores a través de los grupos de almacenamiento, de forma que $\beta_G(\mathbf{S},\mathbf{G})$ toma un valor alto en el caso de que la distribución sea homogénea y toma un valor bajo si es poco uniforme. Si denotamos como N_r al número de servidores que existen en un grupo de almacenamiento G_r y ordenamos los grupos de forma ascendente en función del número de servidores que tienen, la expresión que nos permite calcular el valor del nivel de la homogeneidad de agrupación es la siguiente:

$$N_r = \sum_j S_{r(j)} \qquad\qquad \beta_G(\mathbf{S},\mathbf{G}) = \prod_{i=1,j=i}^{n} N_i/N_j$$

Ejemplo 7.1. Supongamos que tenemos 10 servidores. Se pueden llevar a cabo diferentes agrupaciones. Algunas de estas distribuciones las mostramos a continuación. El factor de agrupación ($\alpha_G(\mathbf{S},\mathbf{G})$) y el nivel de homogeneidad ($\beta_G(\mathbf{S},\mathbf{G})$) aparecen junto a la distribución correspondiente:

Primera distribución:

$$G_1 = \{S_1, S_2, \ldots, S_9, S_{10}\}$$

- $\alpha_G(\mathbf{S},\mathbf{G}) = 10/1 = 10$ (Servidores clones)
- $\beta_G(\mathbf{S},\mathbf{G}) = 10/10 = 1$ (Nivel de homogeneidad máximo: los servidores están distribuidos en la misma proporción a través de los grupos de almacenamiento).

Segunda distribución:

$$G_1 = \{S_1, S_2\}$$
$$G_2 = \{S_3, S_4\}$$
$$G_3 = \{S_5, S_6\}$$
$$G_4 = \{S_7, S_8\}$$
$$G_5 = \{S_9, S_{10}\}$$

- $\alpha_G(\mathbf{S},\mathbf{G}) = 10/5 = 2$

- $\beta_G(\mathbf{S}, \mathbf{G}) = (2/2)^{15} = 1$ (Nivel de homogeneidad máximo: los servidores están distribuidos en la misma proporción a través de los grupos de almacenamiento).

Tercera distribución:

$$G_1 = \{S_1, S_2\}$$
$$G_2 = \{S_3, S_4, S_5\}$$
$$G_3 = \{S_6, S_7, S_8, S_9, S_{10}\}$$

- $\alpha_G(\mathbf{S}, \mathbf{G}) = 10/3 = 3,\hat{3}$
- $\beta_G(\mathbf{S}, \mathbf{G}) = (2/2) * (2/3) * (2/5) * (3/3) * (3/5) * (5/5) = 0,16$

Cuarta distribución:

$$G_1 = \{S_1, S_2\}$$
$$G_2 = \{S_3, S_4\}$$
$$G_3 = \{S_5, S_6, S_7\}$$
$$G_4 = \{S_8, S_9, S_{10}\}$$

- $\alpha_G(\mathbf{S}, \mathbf{G}) = 10/4 = 2,5$
- $\beta_G(\mathbf{S}, \mathbf{G}) = (2/2)^2 * (2/3) *^2 *(2/2) * (2/3)^2 * (3/3)^3 \approx 0,20$

Quinta distribución:

$$G_1 = \{S_1\}$$
$$G_2 = \{S_2\}$$
$$G_3 = \{S_3\}$$
$$G_4 = \{S_4\}$$
$$G_5 = \{S_5\}$$
$$G_6 = \{S_6\}$$
$$G_7 = \{S_7\}$$
$$G_8 = \{S_8\}$$
$$G_9 = \{S_9\}$$
$$G_{10} = \{S_{10}\}$$

- $\alpha_G(\mathbf{S}, \mathbf{G}) = 1$
- $\beta_G(\mathbf{S}, \mathbf{G}) = 1$

- **Nivel de homogeneidad de carga**. Conociendo el nivel de homogeneidad de agrupación, podemos modificar la función de agrupación de forma dinámica para lograr un mayor equilibrado de carga, suponiendo que los servidores reciban la carga de forma homogénea.

  En caso de que la carga no sea homogénea, podemos hablar de un nuevo índice a medir: **nivel de homogeneidad de carga**, que denotaremos como $\gamma_G(S, G)$. El nivel de homogeneidad de carga es equivalente al nivel de homogeneidad de agrupación, en el caso de que cada servidor tenga igual carga de trabajo. En otro caso, para definir el nivel de homogeneidad de carga hay que tener en cuenta un nuevo parámetro: la carga de cada servidor, que denominaremos $l_{r(u)}$ para un servidor $S_{r(u)}$ dado. Este parámetro no es estático, es decir, depende del tiempo ($l_{r(u)} \equiv l_{r(u)}(t)$) y, por tanto, $\gamma_G(\mathbf{S}, \mathbf{G})$ también es dependiente del tiempo ($\gamma_G(\mathbf{S}, \mathbf{G}) \equiv \gamma_G(\mathbf{S}, \mathbf{G}, t)$).

Si denotamos como $L_r(t)$ a la carga de un grupo de almacenamiento en un instante t, se cumple que:

$$\mathrm{L}_r = \sum_j l_{r(j)}$$

En un determinado instante de tiempo t, $\gamma_G(\mathbf{S}, \mathbf{G}, t)$ se define como:

$$\gamma_G(\mathbf{S}, \mathbf{G}, t) = \prod_{i=1, j=i}^{n} L_i(t)/L_j(t)$$

es decir, se calcula de forma similar al nivel de homogeneidad de agrupación, pero teniendo en cuenta la carga del grupo de almacenamiento en lugar del número de servidores del mismo.

- **Granularidad de un grupo de almacenamiento**. Se denomina **granularidad** a la razón existente entre la cantidad de trabajo realizado por una tarea y la cantidad de comunicación que llevó a cabo. En términos genéricos se habla de granularidad *fina* cuando hay poca computación por comunicación, mientras que es *gruesa* cuando hay gran cantidad de computación por comunicación entre tareas. En el caso de un grupo de almacenamiento G_i, se define su granularidad en un determinado intervalo de tiempo T ($\delta_G(G_i, T)$) como la razón entre la carga del mismo y la cantidad de comunicación realizada por el grupo de almacenamiento en dicho intervalo, que denotaremos como $C(G_i)$. Es decir,

$$\delta_G(G_i, T) = (L_{G_i}(t+T) - L_{G_i}(t))/C(G_i)$$

Para definir $C(G_i)$, necesitamos conocer la cantidad de comunicación de cada uno de los servidores que lo forman, es decir:

$$C(G_i) = \sum_j c(S_{i(j)})$$

considerando que $c(S_{i(j)})$ es la cantidad de comunicación llevada a cabo por el servidor $S_{i(j)}$. Este parámetro es difícil de formalizar y depende mucho de la plataforma de distribución utilizada, aunque puede ser medido a través de herramientas de monitorización de la plataforma correspondiente.

El valor de $C(G_i)$ puede ser dividido en coste de comunicación *intra-grupal*, es decir, comunicación debida a los mensajes establecidos por los servidores del grupo G_i y coste de comunicación *inter-grupal*, es decir, comunicación debida a los mensajes establecidos entre los servidores de G_i y cualquier otro servidor de un grupo diferente. En principio, con el uso de los grupos de almacenamiento se tiende a minimizar este segundo término:

$$C(G_i) = C_{intra-grupal}(G_i) + C_{inter-grupal}(G_i)$$
$$C(G_i) = \sum_{j,k} c(S_{i(j)}, S_{i(k)}) + \sum_{j,k,l \neq i} c(S_{i(j)}, S_{l(k)})$$

De la expresión anterior y de las motivaciones de los grupos de almacenamiento, se suponen las siguientes premisas:

- $c(S_i, S_i) = 0$
- Se ha de intentar minimizar el segundo término, es decir: $\sum_{j,k,l \neq i} c(S_{i(j)}, S_{l(k)}) \rightarrow 0$. En el funcionamiento normal del sistema, este término es despreciable frente al valor de $C_{intra-grupal}$, ya que las operaciones son realizadas dentro de un mismo grupo de almacenamiento. En fases de reajuste de grupos (aplicación de las operaciones `mapJoin()`, `mapBreak()` y `mapChange()`), es cuando este factor no es despreciable. Pero estas fases no suelen llevarse a cabo de forma muy frecuente ni intercaladas con las fases de trabajo normal.

- **Esfuerzo de paralelización**. Este factor se puede medir como el tiempo de gestión de las tareas paralelas. El tiempo total de trabajo se puede calcular como la suma del tiempo de gestión y el tiempo de trabajo útil.

- **Nivel de cooperación**. Este factor mide el grado de coordinación que existe entre dos grupos de almacenamiento. Al igual que con la comunicación, existen dos niveles de cooperación: cooperación interna y cooperación externa.

  La cooperación interna es aquélla que existe dentro de un grupo de almacenamiento y tiene sentido debido a que dicho grupo no es una entidad atómica, sino que está formado por un conjunto de MAS (véase sección 5.5). Por tanto, si denominamos $\epsilon_G(G_i)$ al nivel de cooperación interna de un grupo de almacenamiento G_i, se cumple que:

$$\epsilon_G(G_i) = \sum_j \zeta_M(j)$$

  donde $\zeta_M(j)$ mide el nivel de cooperación del sistema multiagente j.

  Denotaremos $\zeta_G(G_i, G_j)$ al nivel de cooperación externa entre dos grupos de almacenamiento G_i y G_j. De este modo, se cumple que:

$$\zeta_G(G_i, G_j, T) = \sum_j \theta_M(i, j)$$

  donde $\theta_M(i, j)$ mide el nivel de cooperación entre dos sistemas multiagente i y j.

- **Capacidad de almacenamiento de un grupo de almacenamiento:** Este parámetro es utilizado para medir la capacidad sobrante de un determinado grupo de almacenamiento. Este factor puede ser útil para determinados procesos, uno de los cuales es el cálculo de la selección del grupo de almacenamiento sobre el cual se va a realizar la redistribución de ficheros, tal y como se describe en la sección 7.4.8. La capacidad de un grupo de almacenamiento G_r se denota como P_r y es igual a la suma de las capacidades de todos los servidores que forman parte del mismo. Al igual que el parámetro "carga de un grupo de almacenamiento", la capacidad de almacenamiento de un grupo es dependiente del tiempo. Por tanto, se cumple que:

$$P_r(t) = \sum_j p_{r(j)}(t)$$

donde $p_{r(j)}(t)$ corresponde a la capacidad de almacenamiento sobrante del servidor $S_{r(j)}$ en el instante t.

Un parámetro relacionado con éste último es la capacidad total de un grupo de almacenamiento G_r, que denominaremos PT_r y que es igual a la suma de las capacidades totales de cada uno de los servidores que posee ($pt_{r(j)}$). Por tanto, la relación existente entre la carga de un grupo de almacenamiento y su capacidad de almacenamiento viene dada por la siguiente expresión:

$$L_r(t) = PT_r(t) - P_t$$

7.7. Políticas de agrupación

De cara a construir la relación de agrupación nos vamos a basar en diferentes políticas, a fin de poder implementar distintos algoritmos de planificación para la construcción de los grupos de almacenamiento.

La relación de agrupación es conocida por el cliente MAPFS, es decir, el cliente es consciente de la relación existente entre los grupos de almacenamiento y los servidores. De hecho, es en la parte cliente donde está almacenada la matriz de relación principal y donde se puede modificar (véase 9.5).

Por otro lado, es posible tener replicados ficheros entre diferentes grupos de almacenamiento, por motivos de tolerancia a fallos o equilibrado de carga. En ese caso, será necesario llevar a cabo una planificación que se encargará de decidir a qué grupo de almacenamiento se envía la operación de E/S correspondiente a un fichero. Además, se puede llevar a cabo una migración de los ficheros a cada grupo de almacenamiento por los mismos motivos.

Por tanto, existen tres parámetros configurables, cuya relación será necesario optimizar mediante el uso de diferentes políticas, a saber:

- Relación de agrupación, es decir, la asociación entre los servidores y los grupos de almacenamiento.
- Asociación entre los ficheros y grupos de almacenamiento.
- Replicación de los ficheros entre diferentes grupos de almacenamiento.

En este trabajo se definen un conjunto de políticas básicas para construir la relación de agrupación principal. No obstante, el sistema puede ser extendido con la adición de nuevas políticas. Las políticas básicas se describen a continuación.

7.7.1. Agrupación por contenido

Los ficheros se distribuyen entre los diferentes servidores de almacenamiento de que disponga el sistema. La distribución de la carga de los servidores puede llevarse a cabo de forma dinámica. Es lógico agrupar a los servidores por su contenido, es decir, por los ficheros que albergan. De este modo, si se optimizan las operaciones por grupo de almacenamiento, se consigue optimizar el acceso a los ficheros. Esta es la política que se utiliza por defecto.

No obstante, tal y como están definidos los grupos de almacenamiento, esta política no permite la replicación de ficheros entre distintos grupos de almacenamiento, porque la relación de agrupación se basa precisamente en el contenido de los servidores. Para eliminar esta restricción podemos hacer uso de otro tipo de grupo de almacenamiento, denominado **grupo de replicación**.

Definición 18 *Denominamos grupo de replicación de un grupo principal o invisible G_x en otro grupo principal o invisible G_y a un grupo invisible G_z, utilizado para almacenar una copia de los ficheros del grupo G_x en los servidores del grupo G_y. Por tanto, se cumple que:*

$$file_u \in S_v \wedge S_v \in G_x \Longleftrightarrow file_u \in S_w / S_w \in G_z$$
$$S_v \in G_y \Longleftrightarrow S_v \in G_z$$

El grupo de replicación G_z es un grupo invisible del grupo principal G_y.

Si redefinimos la política de agrupación por contenido de los servidores de tal forma que se refiera sólo a grupos principales, podemos implementar la replicación a través del uso de los grupos de replicación. Éstos últimos no modifican las propiedades de los grupos de almacenamiento principales, que siguen teniendo una misma relación de agrupación principal, ya que se trata de grupos invisibles. Además por este motivo, la replicación se gestiona de forma transparente a las aplicaciones. De este modo, si queremos replicar ficheros de un grupo de almacenamiento en otro grupo de almacenamiento diferente, crearemos un nuevo grupo de replicación que contiene todos los servidores de este último. A continuación se copiarán los ficheros del primero en el grupo de replicación.

Un grupo de replicación está asociado a dos grupos de almacenamiento diferentes, el grupo con el que comparte los servidores y el grupo con el que comparte los ficheros. Al primero de los grupos se le denomina **grupo proveedor de servidores** y al segundo **grupo proveedor de ficheros**. A ambos grupos se les denomina grupos proveedores de un determinado grupo de almacenamiento.

Definición 19 *Un grupo de almacenamiento G_x se denomina grupo proveedor de servidores de un grupo G_y sii G_y es grupo de replicación de otro grupo G_z en el grupo G_x.*

Definición 20 *Un grupo de almacenamiento G_x se denomina grupo proveedor de ficheros de un grupo G_y sii G_y es grupo de replicación de G_y en otro grupo G_z.*

Las operaciones realizadas sobre los grupos proveedores pueden implicar cambios en el grupo de replicación asociado. La forma en que se deben realizar estos cambios se muestra a continuación:

- Si el grupo proveedor de servidores de un grupo G_x modifica su conjunto de servidores, éste último debe modificar su topología de forma análoga.
- Si el grupo proveedor de ficheros de un grupo G_x modifica alguno de sus ficheros, éste último debe modificar sus correspondientes ficheros de forma análoga.
- La modificación de ficheros de un grupo proveedor de servidores no afecta al grupo de replicación asociado.
- La modificación de la topología de un grupo proveedor de ficheros no afecta al grupo de replicación asociado.

Respecto a la modelización de los grupos de replicación, la partición reticular de los servidores mantiene su estructura. El único cambio que sufre esta estructura es la adición del grupo de replicación entre el grupo proveedor de servidores y el grupo que se encuentra inmediatamente por debajo en el diagrama de Hasse[8]. La relación de pertenencia grupal se mantiene exactamente igual.

Ejemplo 4

[8]El grupo proveedor de ficheros no se utiliza para calcular la partición reticular

Por otro lado, hay que analizar cuál es el comportamiento de la operación de defragmentación `mapGroupDefrag()` frente a los grupos de replicación. Al ejecutar esta operación, los grupos de replicación son los únicos grupos invisibles que no se eliminan, ya que su objetivo es dar soporte a la replicación de ficheros. Por tanto, es necesario distinguir entre grupos de replicación y grupos invisibles en general, de forma que cuando se lleve a cabo la operación de defragmentación, sólo se eliminen aquellos grupos secundarios que no sean además grupos de replicación.

La política de agrupación por contenido permite modificar la relación de agrupación de forma dinámica, siempre que la plataforma distribuida subyacente permita cambiar los ficheros de ubicación física dinámicamente.

7.7.2. Agrupación por topología

Utilizando esta política, los servidores se agrupan según la topología de la red. De este modo, la relación de agrupación se construye en base a la distribución física de los servidores. Una posible opción es agrupar los servidores por cercanía. Esta política permite optimizar las peticiones a un grupo de almacenamiento, basándose en que los mensajes enviados a través de la red a servidores cercanos entre sí, tendrán una latencia similar.

Otra ventaja de esta política es que el sistema multiagente asociado a cada grupo de almacenamiento puede adaptarse perfectamente a éste, debido a la capacidad de los sistemas multiagente de situarse en una determinada topología. Por tanto, el sistema multiagente puede compartir la estructura del correspondiente grupo de almacenamiento, mejorando el tiempo de comunicación existente entre los agentes del mismo.

No obstante, esta política tiene como desventaja el hecho de que establece la relación de agrupación de forma estática. Si se desea modificar la relación de agrupación, es necesario cambiar la topología de la red y a continuación utilizar las operaciones `mapJoin()`, `mapChange()` y `mapDelete()` para establecer las nuevas relaciones existentes entre servidores y grupos de almacenamiento.

7.7.3. Agrupación por similitud

Esta política permite agrupar los servidores por similitud. La similitud se mide evaluando las características técnicas de los servidores. Esta política permite optimizar las peticiones a un grupo de almacenamiento, basándose en el hecho de que todos los servidores que forman dicho grupo tienen una capacidad de procesamiento similar.

Al igual que la política anterior, esta política establece los grupos de forma estática y basándose en aspectos ajenos a la información almacenada. No obstante, tanto esta política como la anterior permiten el uso de prioridades en la asignación de ficheros a los distintos grupos, de forma que se fomente el uso de los grupos de almacenamiento con un valor de prioridad más alto. De este modo, si asignamos a los grupos de almacenamiento con una topología más adecuada o características técnicas mejores, valores más altos de prioridad, favoreceremos el uso de este tipo de grupos, que con seguridad ofrecerán un mejor rendimiento.

7.7.4. Relación entre políticas y parámetros de agrupación

Esta sección muestra cómo afectan las diferentes políticas utilizadas para la construcción de la relación de agrupación a los parámetros de agrupación.

Dependiendo del tipo de política que se defina, los parámetros se pueden interpretar de distintas formas, a saber:

- Si la política utilizada es “agrupación por contenido”, el significado de los parámetros es el

siguiente:

factor de agrupación: se trata del nivel de agrupación que tienen los servidores, teniendo en cuenta que los grupos están formados por servidores que comparten ficheros. Si el factor de agrupación es alto implica que existe una pequeña cantidad de grupos de almacenamiento o un único grupo de almacenamiento. En este último caso, todos los ficheros están distribuidos entre todos los servidores, esto es:

$$file_x \in S_i \ \forall i \in 1, \ldots, n$$

No obstante, puede darse el mismo caso si se da el proceso de "encadenamiento" de ficheros, es decir:

$$\begin{aligned}
&\exists \text{ un conjunto de ficheros } file_i / i \in 1, \ldots, n \text{ tal que:} \\
&file_1 \in S_1 \wedge file_2 \in S_1 \\
&file_2 \in S_2 \wedge file_3 \in S_2 \\
&file_3 \in S_3 \wedge file_4 \in S_3 \\
&\ldots \\
&file_{(n-2)} \in S_{n-2} \wedge file_{(n-1)} \in S_{(n-2)} \\
&file_{(n-1)} \in S_{(n-1)} \wedge file_n \in S_{(n-1)} \\
&file_n \in S_n
\end{aligned}$$

Si el factor de agrupación es bajo implica que existe una gran cantidad de grupos de almacenamiento y, por tanto, no existe una intersección entre los ficheros de cada servidor.

nivel de homogeneidad de agrupación: este parámetro mide la forma en que se distribuye la relación de agrupación. Si el nivel de homogeneidad de agrupación es alto implica que los grupos de almacenamiento tienen una distribución similar en cuanto a número de servidores. En este caso, la compartición de ficheros es homogénea entre los diferentes servidores. Por el contrario, si el nivel de homogeneidad de agrupación es bajo implica que la distribución de los ficheros no es homogénea.

nivel de homogeneidad de carga: El nivel de homogeneidad de carga es un concepto similar al anterior, pero tiene en cuenta que los servidores pueden tener diferente carga. Si el nivel de homogeneidad de carga es alto implica que los grupos de almacenamiento tienen una distribución similar de carga y, por tanto, si el acceso a los ficheros es similar, la eficiencia será mayor y el tiempo de acceso a los ficheros será óptimo. Por el contrario, si el nivel de homogeneidad de carga es bajo y el acceso a los ficheros es similar, algunos servidores serán "cuello de botella" del sistema.

- Si la política utilizada es "agrupación por topología", el significado de los parámetros es el siguiente:

factor de agrupación: se trata del nivel de agrupación que tienen los servidores, teniendo en cuenta que los grupos están formados por servidores dispuestos en una misma topología. Si el factor de agrupación es alto implica que todos los servidores se encuentran en un mismo tramo de una red física y que las latencias de los mensajes enviados son similares. Por el contrario, una red con servidores situados en diferentes tramos físicos conduce a un valor muy bajo de este parámetro.

nivel de homogeneidad de agrupación: Si el nivel de homogeneidad de agrupación es alto implica que los grupos de almacenamiento tienen una distribución similar en cuanto a número de servidores. En este caso, la distribución física de los servidores es homogénea. Por el contrario, si el nivel de homogeneidad de agrupación es bajo implica que la distribución física de los servidores no es homogénea.

nivel de homogeneidad de carga: A diferencia del anterior concepto, éste tiene en cuenta la carga de los diferentes servidores. Si el nivel de homogeneidad de carga es alto implica que los grupos de almacenamiento tienen una distribución similar de carga y, por tanto, el tráfico en la red tendrá una distribución homogénea y el servidor de ficheros dará un servicio eficiente. Por el contrario, si el nivel de homogeneidad de carga es bajo algunos tramos de la red estarán más cargados que el resto y serán "cuello de botella" del sistema.

- Si la política utilizada es "agrupación por similitud", el significado de los parámetros es el siguiente:

factor de agrupación: se trata del nivel de agrupación que tienen los servidores, teniendo en cuenta que los grupos están formados por servidores que son técnicamente similares. En este caso, el factor de agrupación está intimamente relacionado con las características técnicas de los equipos utilizados. Si el factor de agrupación es alto implica que todos los servidores son similares y, por tanto, tienen una capacidad de procesamiento similar. Por el contrario, un valor muy bajo de este parámetro indica diferencias técnicas importantes entre todos los servidores que constituyen el sistema. En este caso es importante redistribuir las peticiones dependiendo del número y características técnicas de los servidores que forman cada grupo de almacenamiento.

nivel de homogeneidad de agrupación: Si el nivel de homogeneidad de agrupación es alto implica que los grupos de almacenamiento tienen una distribución similar en cuanto a número de servidores. En este caso, hay un número similar de servidores con diferentes características técnicas. Por el contrario, si el nivel de homogeneidad de agrupación es bajo implica que la distribución de los servidores respecto a sus capacidades de procesamiento no es homogénea. En este último caso, interesa estudiar de nuevo el número y características técnicas de los servidores que forman cada grupo de almacenamiento, a fin de redistribuir las peticiones.

nivel de homogeneidad de carga: Si el nivel de homogeneidad de carga es alto implica que los grupos de almacenamiento tienen una distribución similar de carga y, por tanto, la carga es independiente de la capacidad de procesamiento. Esto puede no interesar en el caso de que haya muchas diferencias de un grupo a otro. Lo ideal es que los servidores con mayor carga sean aquéllos con mayor capacidad de procesamiento. Por otro lado, si el nivel de homogeneidad de carga es bajo, es posible que haya servidores no utilizados y otros sobrecargados y que, por tanto, constituyan el "cuello de botella" del sistema.

Los restantes parámetros no están relacionados con la política utilizada, sino con las características de los servidores y el modelo de cooperación e implementación del sistema de ficheros MAPFS.

7.8. Nombrado de ficheros

Como vimos en la sección 5.4.3, MAPFS utiliza un esquema de nombrado basado en los sistemas de ficheros tipo UNIX, pero en el caso de MAPFS hay que extender dicho concepto para que permita el

manejo de grupos de almacenamiento. Ahora que conocemos los grupos de almacenamiento, estamos en condiciones de especificar esta extensión.

En cada uno de los nodos se almacenan los ficheros correspondientes al grupo de almacenamiento al que pertenecen.

El nombre del fichero es un nombre global, igual para cada uno de los nodos en los que se almacena. El sistema de directorios tiene la siguiente estructura:

> /mapfs/group<*id_stg_group*>/<*filename*>
> donde *id_stg_group* corresponde al identificador del grupo de almacenamiento y
> *filename* es el nombre del fichero global

Siguiendo esta estructura, se puede utilizar otro concepto básico del sistema de ficheros tipo UNIX, el *montaje*. Se puede utilizar como punto de montaje del grupo de almacenamiento <*i*> cualquier directorio y como directorio de almacenamiento en el servidor */mapfs/group*<*i*>, de modo que se monten de forma transparente al usuario cada uno de los directorios */mapfs/group*<*i*< de los nodos servidores sobre el directorio especificado por el usuario. Las operaciones se realizarán contra dicho grupo de almacenamiento.

La figura 7.8 muestra la forma de llevar a cabo el proceso de montaje mediante el uso de grupos de almacenamiento. La figura representa el proceso, mostrando cada una de las capas de la arquitectura de MAPFS y estableciendo el orden en el que se llevan a cabo cada uno de los subprocesos.

Como se puede observar en la figura, este proceso está formado por un conjunto de pasos:

1. El cliente MAPFS se encarga de "montar" un determinado grupo de almacenamiento, sobre el cual va a realizar las operaciones de E/S. Este grupo de almacenamiento se montará sobre un determinado directorio. El directorio donde están almacenados los ficheros MAPFS en los servidores es *mapfs/group*<*i*> para el grupo principal cuyo identificador sea *i*. La interfaz del subsistema MAPFS_FS proporciona la operación de montaje, que tiene la siguiente estructura:

   ```
   void mapMount (Mapfs_Group group, char *dir);
   ```

2. Para realizar el montaje, se debe acceder a la base de datos de grupos[9], que contiene la información sobre los nodos que forman el grupo de almacenamiento. El sistema multiagente se encarga de implementar este paso. En el caso de que el fichero esté almacenado en un grupo principal, la matriz de agrupación contiene esta información.

3. Una vez conocida la configuración, se puede acceder a los nodos en los que se distribuye el fichero.

4. Una vez conocido el nodo, se puede acceder al directorio *mapfs/group*<*i*> de dicho servidor.

5. Las operaciones de E/S se realizan de forma transparente al usuario sobre los ficheros almacenados en los nodos servidores. Dependiendo de la función de distribución utilizada, los datos se distribuyen de distinto modo entre los nodos existentes en dicho grupo de almacenamiento (véase sección 7.9.

Esta es la operativa para los grupos de almacenamiento principales. Para los grupos secundarios y de replicación el nombrado es diferente. A los grupos secundarios se les añade el sufijo "_s" y el número

[9]Se describe en la sección 9.5.1

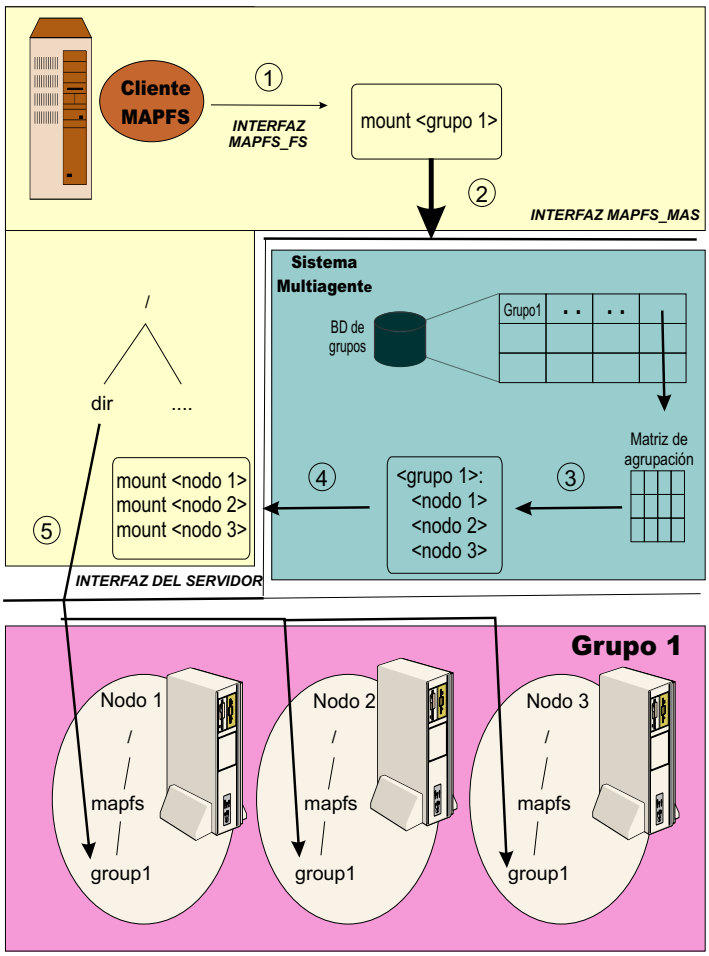


Figura 7.8: Proceso de montaje en MAPFS

del grupo secundario, de forma que los ficheros pertenecientes al grupo secundario número 3 del grupo principal 2 se almacenan en la ruta */mapfs/group<2>_s3* de cada uno de los servidores. De forma análoga, a los grupos de replicación se les añade el sufijo "_r" y el número de grupo de replicación, de tal forma que los ficheros pertenecientes al grupo de replicación número 1 del grupo principal 3 se almacenan en la ruta */mapfs/group<3>_r1*. El proceso de montaje es análogo al de los grupos principales, salvo que no se utiliza la matriz de agrupación para obtener los nodos correspondientes.

De este modo, pueden existir ficheros con el mismo nombre en distintos grupos de almacenamiento. El nombrado es transparente al usuario. La aplicación de usuario sólo debe especificar el punto de montaje. A través de las tablas de la base de datos de grupos se puede conocer si el fichero se encuentra en un grupo principal o en un grupo secundario. De forma transparente, también se puede acceder a posibles réplicas en otros grupos de almacenamientos.

Por último, cuando un cliente no vaya a utilizar un determinado grupo de almacenamiento, es conveniente que realice una operación que "desmonte" el grupo de almacenamiento. Esta operación tiene el siguiente formato:

```
void mapUmount (char *dir);
```

Después de llevar a cabo esta operación, el directorio *dir* no estará asociado al grupo de almace-

namiento correspondiente.

Las ventajas que ofrece el montaje de ficheros MAPFS son principalmente dos:

1. Permite ocultar el uso de grupos de almacenamiento. Una vez montado el grupo, las aplicaciones acceden a los ficheros pertenecientes a dicho grupo a través del punto de montaje.

2. Homogeneidad con otros sistemas de ficheros, como el de UNIX, que utilizan el montaje para acceder a diferentes dispositivos del sistema.

7.9. Funciones de distribución

Las funciones de distribución determinan cómo distribuir la información a través de los distintos nodos que forman un grupo de almacenamiento. La distribución de los datos proporciona la capacidad al sistema para realizar las operaciones de E/S de una forma paralela. Por este motivo, la distribución y, por extensión, las funciones de distribución toman un papel fundamental en MAPFS.

Una función de distribución asocia un determinado *offset* o desplazamiento de la información con un nodo en el grupo de almacenamiento.

Definición 21 *Se denomina función de distribución a una función* $fd_G(G_x, file, offset) = (S_y, O_y)$, *donde:*

- G_x*: Identificador del grupo de almacenamiento en el cual se va a realizar la distribución.*
- *file: Nombre del fichero al que se va a acceder.*
- *offset: Desplazamiento de la información.*
- S_y*: Identificador del nodo al que corresponde dicho desplazamiento.*
- O_y*: Desplazamiento de la información correspondiente al nodo.*

De cara a optimizar el rendimiento global del sistema, el criterio que se suele utilizar para definir diferentes funciones de distribución es maximizar la distribución para favorecer el paralelismo en las operaciones de acceso a los datos. Es por ello que la función *round-robin*, que permite distribuir los datos de una forma circular, dependiendo del tamaño de bloque elegido, puede proporcionar unos buenos parámetros de distribución. La elección del tamaño de bloque es un aspecto clave en el rendimiento de esta función de distribución, ya que si seleccionamos un tamaño de bloque pequeño lograremos maximizar la distribución entre los nodos, pero se incrementará el número de operaciones de E/S, ya que los datos están más fragmentados entre los diferentes nodos. Por otro lado, si elegimos un tamaño de bloque grande, los datos pueden no distribuirse homogéneamente entre los servidores[10] (a no ser que el tamaño del fichero sea un múltiplo del tamaño de bloque), pero se decrementará el número de operaciones de E/S. Por tanto, es necesario llegar a un compromiso entre ambas opciones. Además, dependiendo del tamaño de los ficheros utilizados por el sistema, la elección del tamaño de bloque también puede variar.

De este modo, y de una forma operativa, en el caso de utilizar *round-robin*, los parámetros que nos interesa calcular a la hora de llevar a cabo la distribución en un grupo de almacenamiento compuesto por servidores vacíos a partir de los datos proporcionados por el usuario a las operaciones de E/S son:

- *Nodo* o nodos en el que debe almacenarse/obtenerse la información. Para ello se utiliza el nodo inicial, descrito en la sección 7.4.2. Una vez almacenados los datos en este nodo inicial, el resto se almacena de forma circular.

[10] desequilibrando el nivel de homogeneidad de carga

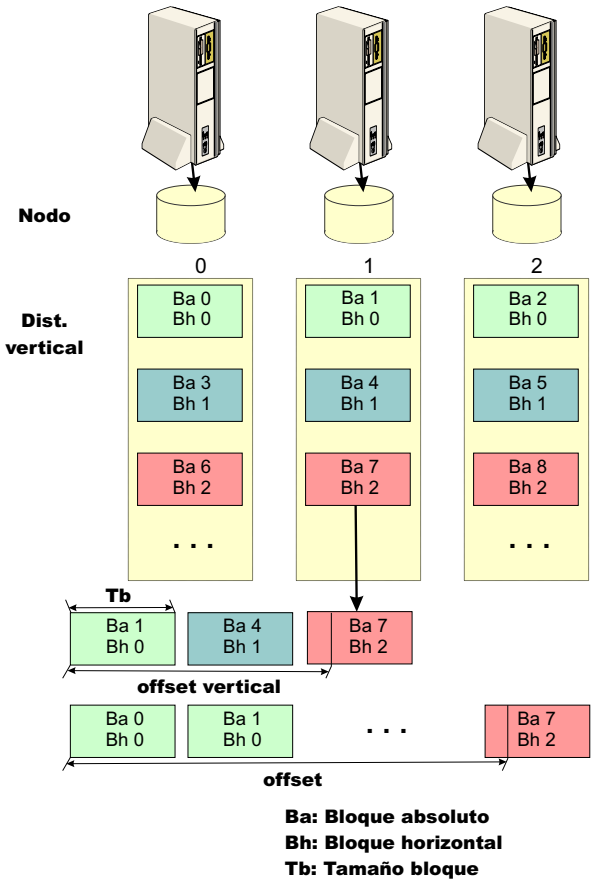


Figura 7.9: Distribución de los datos en el sistema de ficheros MAFPS

- *Bloque absoluto:* Se trata del bloque de un fichero visto de una forma global, del fichero completo, es decir, como si no se hubiera distribuido.
- *Bloque horizontal:* Se trata del concepto de bloque desde el punto de vista de la distribución horizontal del mismo sobre los diferentes nodos.
- *Offset vertical:* Se trata del desplazamiento de un fichero visto desde el punto de vista de un nodo. Se calcula a partir del *offset* global[11].

La figura 7.9 muestra el proceso de distribución suponiendo que en un determinado grupo de almacenamiento tenemos tres servidores y que el nodo inicial es el nodo 0. El esquema sería equivalente en caso de que el nodo inicial fuera diferente, salvo que los bloques se mostrarían desplazados respecto al nodo inicial.

Las expresiones que permiten calcular los parámetros anteriores se enuncian a continuación y requieren conocer el tamaño del bloque del sistema (*tBloque*), el número de servidores del grupo de almacenamiento (*N_SERV*) y el *offset global* (que de forma simplificada denominaremos *offset*) a partir del cual se realizara la operación de E/S.

- *Nodo (nodo):* El nodo inicial a partir del cual se van a distribuir o están distribuidos los datos de un determinado fichero se calcula a partir de un algoritmo que tiene como salida dicha

[11] El offset global es proporcionado a través de las primitivas del sistema de ficheros

información y que está descrito en la sección 7.4.2. Los datos se van distribuyendo en un orden circular a través de los nodos del grupo de almacenamiento. Por tanto:

$$\text{nodo} = (\text{nodo\_anterior}+1)\ \%\text{N\_SERV}$$

donde % representa la operación módulo. A esta **función de distribución** se le denomina *round-robin*, por analogía con el algoritmo de planificación con el mismo nombre [Tan87], [Dei87]. Se puede utilizar cualquier otra función de distribución. Este tipo de funciones junto con las políticas a la hora de crear los grupos de almacenamiento son configurables y van a condicionar el rendimiento del sistema (véase secciones 7.7 y 7.9).

- *Bloque absoluto (bAbsoluto):* Determinar el bloque absoluto consiste en calcular el número de orden que ocupa el bloque que contiene el dato representado por *offset* en el fichero global o completo. Por tanto:

$$\text{bAbsoluto} = \text{offset}/tBloque$$

- *Bloque horizontal: (bHorizontal)* Determinar el bloque horizontal consiste en calcular el número de orden que ocupa el bloque que contiene el dato representado por *offset* respecto a la distribución horizontal del fichero a través de los nodos. Por tanto:

$$\text{bHorizontal} = \text{bAbsoluto}/\text{N\_SERV}$$

- *Offset vertical (oVertical):* El *offset vertical* representa el desplazamiento de un dato a través de un único nodo. Por tanto:

$$\text{oVertical} = (bHorizontal * tBloque) + offset\,\%tBloque$$

Conociendo el nodo y el *offset vertical*, conocemos el lugar exacto donde se ubica un determinado dato que se desea leer o escribir.

Por tanto, la función de distribución *round-robin* se puede definir:

$$fdRR_G(G_x, file, offset) = (S_y, O_y)$$
$$fdRR_G = ((beginningNode(file, StorageCapList) + (offset/tBloque))\ \%\text{N\_SERV}, \text{oVertical})$$

considerando que los nodos del grupo de almacenamiento se numeran desde el 0 hasta *N_SERV*.

Como podemos ver, la elección del tamaño de bloque (*tBloque*) condiciona la distribución *round-robin*.

El sistema MAPFS permite la definición de otras funciones de distribución diferentes, las cuales serán utilizadas por las operaciones de E/S a la hora de acceder a la información.

7.10. Trabajos relacionados

Existen algunos sistemas de ficheros que tienen conceptos similares a los grupos de almacenamiento de MAPFS. En concreto, el sistema de ficheros xFS utiliza el concepto de ***stripe group*** para definir subconjuntos de servidores de almacenamiento. Por su parte, GFS denomina **NSP** (*Network Storage Pool*) a una colección de dispositivos compartidos físicamente. A continuación se describen las diferencias y analogías de cada una de estas propuestas con la de MAPFS.

7.10.1. *Stripe group* en XFS

En xFS, un *stripe group* es un conjunto de servidores de almacenamiento. xFS distribuye los datos a través de los discos de los servidores de almacenamiento, implementando un RAID software. Además utiliza un sistema de *striping*, basado en *logs*, de forma análoga al sistema Zebra. Los *stripe groups* tienen como funcionalidad básica eliminar el problema de la escalabilidad, que surge cuando el *stripe* abarca a un gran número de discos. Llevando a cabo esta agrupación, es posible realizar el proceso de distribución sin que el rendimiento se vea decrementado. De hecho, Zebra, que no utiliza estas agrupaciones, queda limitado precisamente por un número máximo de servidores que puede utilizar de una forma eficiente. Al igual que en MAPFS, xFS permite llevar a cabo una reconfiguración dinámica, cuando un nodo deja de funcionar o se une a xFS. Para gestionar el funcionamiento de los *stripe groups*, xFS utiliza una estructura denominada *stripe group map*, que recoge información sobre cada *stripe group*. De cara a permitir la reconfiguración, xFS distingue dos tipos de grupos, grupos actuales y grupos obsoletos. Cuando un servidor de almacenamiento entra o deja de formar parte del sistema, xFS cambia el *map*, de forma que cada servidor activo pertenezca exactamente a uno de los grupos de almacenamiento actuales. Si esta reconfiguración cambia la membresía de un grupo concreto, xFS no borra la antigua entrada del grupo. En su lugar, marca dicha entrada como "obsoleta". Los clientes sólo escriben sobre grupos actuales, aunque pueden leer tanto de grupos actuales como obsoletos. De este modo, no se hace una transferencia de datos de los grupos obsoletos a los grupos actuales, evitando degradar el funcionamiento del sistema. En los sistemas basados en *logs*, existe un proceso que se encarga de ir eliminando las entradas obsoletas. En este caso, un proceso *cleaner* es el encargado de transferir los datos desde los grupos obsoletos a los grupos actuales a través del tiempo. Cuando este proceso termina de mover el último dato de un grupo obsoleto, xFS elimina su entrada del *stripe group map*.

Por tanto, las similitudes existentes entre los *stripe groups* de xFS y los grupos de almacenamiento de MAPFS son:

1. Ambos sistemas permiten una reconfiguración dinámica. El concepto de grupo actual de xFS equivale a grupo principal de MAPFS y el de grupo obsoleto a grupo secundario de MAPFS.

2. La estructura *stripe group map* es equivalente a la base de datos de grupos de MAPFS.

Por su parte, las diferencias entre ambas propuestas son:

1. xFS implementa los *stripe groups* para poder llevar a cabo RAID sobre un gran número de discos, sin afectar al rendimiento, tal y como se propone para sistemas de RAID grandes [CLG+94]. MAPFS permite la definición de diferentes políticas de agrupación, en función de los parámetros que se deseen optimizar.

2. Los grupos de almacenamiento de MAPFS constituyen un formalismo, que se basa en conceptos matemáticos tales como la partición y los retículos.

3. Los grupos obsoletos de xFS son grupos de sólo lectura, frente a los grupos secundarios, que son de lectura y escritura. Esto es debido al mecanismo de *log* utilizado por xFS.

4. La eliminación de grupos obsoletos en xFS se realiza a través del *cleaner*, proceso utilizado en xFS de forma genérica, frente a la eliminación de grupos secundarios en MAPFS, que tiene que realizarse de forma explícita a través de la operación de defragmentación.

5. MAPFS permite la replicación de los ficheros mediante el uso de grupos de almacenamiento de replicación.

7.10.2. NSP en GFS

En GFS, se denominan NSP o *pools* a las colecciones de dispositivos físicamente compartidos. Los *subpools* permiten particionar los NSP de acuerdo a los atributos o características de dichos dispositivos. Estas características corresponden a la latencia y al ancho de banda de los mismos. Un *subpool* de dispositivos de alto ancho de banda contiene dispositivos conectados a los clientes a través de uno o más enlaces que permiten proporcionar esta característica. Por su parte, un *subpool* de baja latencia está formado por dispositivos de estado sólido. Una implementación de GFS puede explotar diferentes características de rendimiento, utilizando diferentes *subpools*. Por ejemplo, puede situar ficheros frecuentemente referenciados en *subpools* de baja latencia y grandes ficheros en *subpools* de alto ancho de banda. Colocando datos y metainformación en diferentes *subpools* puede también incrementar el rendimiento de las operaciones de E/S, colocando, por ejemplo, la metainformación en *subpools* de baja latencia y los datos reales de los ficheros en *subpools* de alto ancho de banda.

Por otro lado, se define en GFS los grupos de recursos o RG (*Resource Groups*). Los RG distribuyen los recursos del sistema de ficheros a través de un NSP. Pueden existir múltiples RG por dispositivo. Los RG facilitan la colocación de los ficheros en los diferentes *subpools*.

Usuarios avanzados o determinadas aplicaciones pueden explotar el paralelismo transferiendo ficheros entre los RG. La migración de ficheros permite repartir la carga entre los diferentes dispositivos.

Los datos y la metainformación pueden proyectarse sobre múltiples grupos de recursos y *subpools*.

Por tanto, las similitudes existentes entre los *pools* de GFS y los grupos de almacenamiento de MAPFS son:

1. La política de agrupación por similitud es análoga a la forma en que se dividen los NSP en *subpools*.

2. Ambas propuestas permiten realizar una migración de los ficheros, a fin de equilibrar la carga del sistema global.

Por último, las diferencias entre ambas técnicas son:

1. Los grupos de almacenamiento MAPFS permiten agrupar servidores de almacenamiento, frente a los *pools*, que consisten en agrupaciones de dispositivos físicos.

2. Los grupos de almacenamiento permiten definir distintas políticas de agrupación.

3. Los datos se proyectan sobre un único grupo de almacenamiento principal o secundario, aunque es posible tenerlos replicado en los grupos de replicación.

4. En GFS no se define una reconfiguración dinámica de los grupos.

7.11. Resumen

Este capítulo ha descrito el concepto de grupo de almacenamiento, enumerando las ventajas que ofrece y describiendo las características del mismo.

Esta sección muestra cómo la introducción de los grupos de almacenamiento permite dotar al sistema de las características que se enumeraban en la introducción del capítulo, a saber:

1. **Abstracción lógica del concepto de servidor:** Los grupos de almacenamiento permiten abstraer de forma lógica el concepto de servidor, ya que en el sistema de ficheros MAPFS las operaciones referencian a los grupos de almacenamiento en lugar de referenciar a los nodos y éstos simplemente quedan como concepto intermedio entre los ficheros y los grupos de almacenamiento.

2. **Gestión dinámica de los servidores:** El sistema permite gestionar de forma dinámica los nodos dentro de los grupos de almacenamiento, a través de operaciones tales como la unión de un servidor a un grupo de almacenamiento, unión de dos grupos de almacenamiento o eliminación de un nodo de un grupo de almacenamiento.

3. **Eficiencia de las operaciones de almacenamiento:** El uso de diferentes políticas para optimizar los grupos de almacenamiento hace que el acceso a los mismos se realice de una forma más eficiente.

4. **Reparto de carga:** Debido a la facilidad que ofrecen los grupos de almacenamiento para calcular la carga de un determinado grupo, se puede hacer un equilibrado de la carga para permitir que el uso del sistema sea más eficiente.

5. **Migración transparente:** Las operaciones de grupos permiten migrar la información entre distintos grupos de almacenamiento, siguiendo determinados criterios basados en el cálculo de parámetros.

Por tanto, el uso integrado de los grupos de almacenamiento y el sistema de ficheros MAPFS proporciona mayor flexibilidad, rendimiento y funcionalidad a las operaciones de E/S. Y deducimos que la solución dada por los grupos de almacenamiento ofrece la característica de sinergia, que guió la búsqueda en este capítulo.

Capítulo 8

Optimizaciones de MAPFS

8.1. Introducción

El sistema MAPFS proporciona una interfaz que permite realizar E/S colectiva. No obstante, esto no es suficiente para adaptar las necesidades de las aplicaciones al sistema de almacenamiento que le da soporte.

Los modelos de E/S dentro del sistema MAPFS surgen como respuesta a los siguientes problemas:

- Las aplicaciones tienen diferentes patrones de acceso a los datos, dependientes del dominio de la propia aplicación.
- La información adicional que permite optimizar el rendimiento de acceso a los datos es también dependiente de la aplicación.
- La mayoría de los patrones de acceso estudiados corresponden a aplicaciones científicas. De hecho, se han llevado a cabo una gran cantidad de estudios sobre cargas de trabajo de E/S científicas, tanto en entornos paralelos como secuenciales, entre los que destacan [Cro89], [MK91b], [PP93], [Kot93], [CK93], [CHKM93], [GGL93] y [dRC94].

8.2. Optimizaciones de MAPFS

A pesar de que MAPFS puede ser visto como un sistema de ficheros de propósito general, se puede configurar a fin de ajustar su funcionamiento y rendimiento a diferentes **patrones de acceso** de E/S. En el entorno de los sistemas de ficheros, los patrones de acceso se utilizan para incrementar el rendimiento de las operaciones de E/S.

Con el objetivo de optimizar el sistema de E/S, los sistemas de ficheros tienen determinados parámetros configurables que permiten adaptar la funcionalidad de los mismos a las aplicaciones que los utilizan. Los principales parámetros de configuración del sistema de ficheros MAPFS son:

- ***Caching* y *Prefetching* de E/S**: Los datos procedentes del disco y cuya probabilidad de uso sea alta son almacenados en memoria cache, de forma que el tiempo de acceso a ellos sea menor.

Esta técnica permite incrementar el rendimiento de las operaciones de E/S. Esta operación se realiza en MAPFS mediante MAPFS_MAS, que como hemos visto, es un subsistema multiagente independiente. Este subsistema puede configurarse y permite realizar un ajuste del sistema dependiendo de los patrones de acceso de E/S. Las características androides de los agentes que lo forman (inteligencia, proactividad y autonomía) permiten reajustar los parametros de *caching* y sus políticas de una forma dinámica.

- ***Hints***: La tarea anterior puede realizarse de una forma más eficiente si se utilizan *hints* o pistas sobre los patrones de acceso futuros. Los sistemas de almacenamiento que utilizan *hints* permiten mejorar el rendimiento del sistema debido a que proporcionan información que logra decrementar los fallos de cache y llevar a cabo un *prefetching* de los datos que más probablemente sean utilizados en las proximas ejecuciones. En otras palabras, cuanta mayor información tenga el sistema sobre los accesos, menor incertidumbre habrá a la hora de "adivinar" los accesos futuros y, por tanto, los resultados del *prefetching* y *caching* mejorarán. Los *hints* también permiten optimizar las operaciones de E/S en combinación con otras técnicas diferentes, debido a que se encargan de proporcionar información adicional sobre la información almacenada.

Además de estas características, como la mayoría de los sistemas de ficheros, MAPFS posee algunas características que permiten incrementar el rendimiento y la flexibilidad de las aplicaciones que utilizan el API de MAPFS. Como se vió en la sección 5.4, el API de MAPFS ofrece diferentes alternativas: accesos no contiguos (tanto en memoria como en el fichero)[1], E/S colectiva[2], E/S no bloqueante[3] y la posibilidad de comprobar si una operación de lectura o escritura no bloqueante se ha completado[4]. Estas características junto con las anteriores permite que MAPFS se adapte a las necesidades de diferentes aplicaciones.

La mayoría de los sistemas de ficheros ofrecen transparencia total a las aplicaciones de usuario, ocultando detalles sobre la distribución o *layout* de los datos. Algunos sistemas de ficheros tales como Vesta [CF96], [CFPS93], [CF94] o nCUBE [DdR91], [dR92b], [dR92a] proporcionan un control explícito del usuario sobre la forma en que los datos son almacenados, a fin de optimizar los accesos *esperados* sobre los datos. La diferencia fundamental que existe entre las optimizaciones de MAPFS y este tipo de sistemas de ficheros es que MAPFS proporciona una interfaz para que la aplicación de usuario pueda especificar el acceso a los datos. Para ello, la aplicación debe basarse en la semántica de uso de dicha información. Esta diferencia es muy sutil, pero muy importante, ya que permite una mayor flexibilidad.

A continuación se van a analizar las dos capacidades del sistema MAPFS anteriormente mencionadas.

8.3. *Caching* de E/S y *prefetching*

Para utilizar estas características, es necesario tener una estructura cache, de forma que determinados datos de E/S se almacenen en memoria.

La eficiencia de una cache está determinada en gran medida por la política de reemplazo. Una política de reemplazo muy utilizada es la política **LRU** (*Least Recently Used*). Esta política asume que los datos utilizados más recientemente (basándose en la localidad temporal[5]) son probablemente

[1] por ejemplo, operación *mapReadStrided()*
[2] por ejemplo, operación *mapReadStridedColl()*
[3] por ejemplo, operación *mapIReadStrided()*
[4] por ejemplo, operación *mapReadDone()*
[5] La localidad temporal establece que si un elemento ha sido referenciado en un determinado instante, muy probablemente será referenciado en un tiempo cercano

utilizados nuevamente y, por tanto, cada nuevo acceso a un elemento mueve el mismo a la cola de la lista de elementos de dicha cache. El elemento que se encuentra al frente de la lista es el primer candidato a abandonar la misma.

Otras políticas ampliamente utilizadas son la política FIFO (*First In First Out*), que reemplaza el elemento más antiguo o la MRU (*Most Recently Used*), también conocida como LIFO, que reemplaza el elemento más recientemente utilizado.

Frecuentemente se han utilizado históricos para los procesos de *caching* y *prefetching*. De hecho, la política LRU es un tipo específico de histórico, en el cual la historia consiste en los accesos más recientes a los datos. En el caso de prefetching, la técnica más utilizada es *"sequential readahead"*, que consiste en realizar lecturas secuenciales por adelantado, basándose en el hecho de que estas lecturas serán utilizadas en un futuro cercano con una alta probabilidad. Esta técnica es utilizada en la mayoría de los sistemas operativos [FO71], [MJLF84] que explotan la preponderancia de lecturas secuenciales de ficheros completos [OCH+85], [BHK+91]. No obstante, dicha técnica tiene una utilidad limitada cuando los ficheros son pequeños y es incluso contraproducente cuando los patrones de acceso son no secuenciales.

8.4. Incremento del rendimiento a través del uso de *hints*

MAPFS utiliza *hints* en el acceso a los ficheros. Los *hints* se definen como aquellas estructuras conocidas y construídas por el sistema de ficheros y que permiten mejorar el rendimiento de las rutinas de lectura y escritura.

Los *hints* se almacenan dentro del *map-node* definido en la sección 5.4. En MAPFS, los *hints* se pueden determinar de dos formas diferentes:

- Los *hints* pueden ser proporcionados por el usuario, es decir, la aplicación de usuario proporciona al sistema de ficheros la información necesaria para incrementar el rendimiento de las rutinas de E/S sobre un determinado fichero.
- El subsistema multiagente MAPFS_MAS puede construir los *hints*. De hecho, una de las características de los agentes es su capacidad de **aprendizaje**. La creación de *hints* de forma dinámica es posible debido a esta característica. Si se selecciona esta opción, el sistema multiagente debe estudiar los patrones de acceso de la aplicación correspondiente a fin de construir los *hints* que permitan optimizar los procesos de *caching* y *prefetching*.

Los *hints* dependen del patrón de acceso a los datos de las aplicaciones. A este patrón o perfil de acceso le denominaremos **esquema de acceso de datos**. El análisis de los perfiles de acceso es una de las estrategias más utilizadas para lograr soluciones de alto rendimiento a problemas de diferentes dominios. En base a este concepto, se definen los *hints* como la translacción de los esquemas de acceso de datos a la estructura de *hints* del *map-node* (véase sección 5.4).

Como ejemplo de uso de *hints* del sistema de ficheros MAPFS, se va a considerar una aplicación de *Data Mining* [PPG+02b], que permite obtener patrones de comportamiento de conjuntos de datos que se almacenan en bases de datos. En este tipo de escenarios, los esquemas de acceso de datos se basan en estructuras de tipo vector, debido a que las tablas que se utilizan en *Data Mining* tienen esta estructura. Las operaciones de lectura y escritura se realizan por filas. De este modo, si un elemento de la tabla es leído o escrito, es muy probable que la fila que contiene el elemento sea leída o escrita en sucesivas ejecuciones. Por tanto, de cara a optimizar las operaciones de lectura y/o escritura, se puede proceder a almacenar en cache o realizar un *prefetching* de la fila entera (véase figura 8.1).

Este esquema de acceso puede ser modelado y resuelto a través del particionado de datos MPI-IO [CFF+95], concretamente a través del uso de un tipo de datos *vector*.

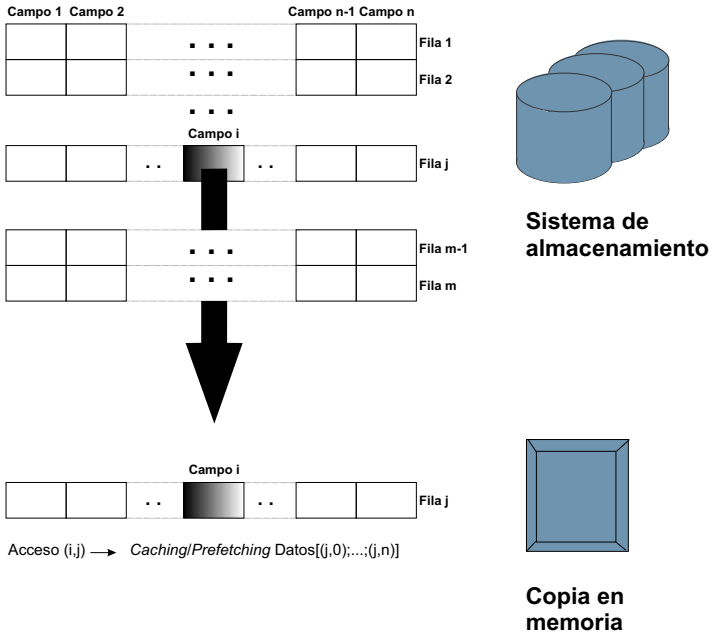


Figura 8.1: *H*ints para realizar el *c*aching y *p*refetching de datos utilizando una estructura tabular

Pero si dos tablas están relacionadas, la regla para construir los *hints* adecuados es un poco más complicada, ya que es necesario aplicar la operación *join* sobre los datos. La figura 8.2 muestra el resultado de construir los *hints* para el procesamiento de dos tablas relacionadas por un atributo o campo.

Si existen más interacciones entre las tablas, las reglas son más complejas. La semántica de las aplicaciones que utilizan un sistema de *Data Mining* afecta a los *hints* del sistema de E/S subyacente. Por tanto, este último sistema debe ser suficientemente flexible para permitir que las aplicaciones puedan modificar los *hints*. En este caso, los *hints* se deben utilizar para optimizar el acceso a la base de datos, ya que esta operación tiene una alta carga computacional. El caso óptimo es aquél en el que sólo se accede a los registros seleccionados.

La mayoría de los accesos a los datos pueden establecerse en base a **atributos**. En el caso de aplicaciones de *Data Mining*, las consultas o *queries* se basan en los valores de determinados atributos de la base de datos. En otro tipo de aplicaciones, se puede utilizar metainformación que, como su propio nombre indica, almacene información adicional sobre los propios datos, permitiendo acceder de una forma más eficiente a los mismos. Esta metainformación también puede estructurarse en base a atributos. Por tanto, sería deseable que los *hints* en MAPFS fueran capaces de almacenar expresiones regulares sencillas, formadas por operadores booleanos aplicados a dichos atributos. Estas expresiones permiten clasificar la información almacenada. En el caso de aplicaciones de *Data Mining*, permiten analizar las diferentes tuplas de la base de datos y evaluarlas. Estas expresiones deben cumplir al menos dos requisitos:

1. Deben ser almacenadas como metainformación asociada a los datos.

2. Deben ser calculadas en poco tiempo, debido a que su objetivo principal es incrementar el rendimiento del sistema.

Por estos motivos, es necesario almacenar todas estas reglas y expresiones en la estructura de *hints*

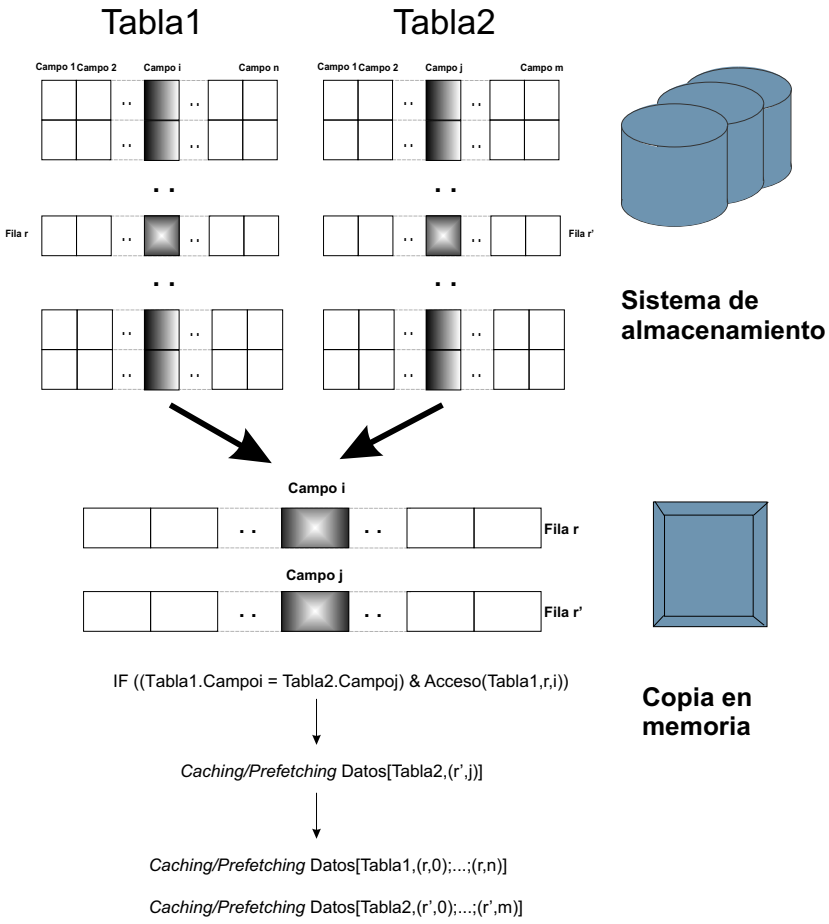


Figura 8.2: *H*ints para realizar el *c*aching y *p*refetching de datos utilizando dos estructuras tabulares

y, por tanto, debe definirse una sintaxis apropiada para este almacenamiento. La próxima sección describe dicha sintaxis.

Como ejemplos de aplicaciones científicas, se pueden considerar operaciones sobre matrices. Desde el punto de vista del uso de los *hints*, la suma de dos matrices es equivalente a la operación sobre dos estructuras tabulares, representada en la figura 8.2. En este caso, al acceder al primer elemento de la fila `i` de la primera matriz, se podría optimizar la lectura de las matrices si se llevara a cabo un *prefetching* de las filas `i` completas de ambas matrices. El problema aparece cuando una fila no cabe de forma completa en la cache. En este caso, la regla no es tan obvia. Este ejemplo se analiza en la sección 8.4.1.

En el caso de la multiplicación de dos matrices, las reglas para realizar un *prefetching* óptimo tampoco son tan obvias.

Supongamos una operación de multiplicación entre matrices:

$$A[M,N] * B[N,P] = C[M,P]$$

tal que las matrices están almacenadas por filas en sus respectivos ficheros y son de un tamaño considerable, tal que incluso una fila o columna no cabe de forma completa en la cache.

Por otro lado, supongamos un algoritmo de multiplicación tradicional:

```
for (i=0; i<M; i++)
        for (j=0; j<P; j++)
                for (k=0; k<N; k++)
                        C[i][j]+=A[i][k]*B[k][j];
```

A continuación, se analizan las posibles estrategias de *prefetching*.

8.4.1. Reglas de *prefetching* sobre la matriz A

Con respecto a la matriz A, en lugar de leer por anticipado la fila correspondiente al elemento pedido, convienen otras estrategias, debido a los siguientes hechos:

- Cada vez que se acaba de trabajar con la fila en curso (cada vez que avanza "i" después de usar la fila P veces), queda un hueco en el *prefetching*. En este caso, sería conveniente pedir por anticipado elementos de la fila "i+1".
- Es posible que no exista espacio suficiente para almacenar la fila completa en la cache. Luego, sería razonable leer anticipadamente sólo una ventana de K valores, que corresponderían a los K elementos siguientes al pedido, pero de una forma circular (módulo N), donde K puede venir determinado por distintos factores, tales como el tamaño de la cache, la carga del sistema o incluso puede ser dinámico.

Según esto, se pueden definir las siguientes reglas:

- **REGLA 1:**
  Se puede proponer un prefetching de fila con ventana circular K (módulo N), un ciclo de P repeticiones por fila, un avance secuencial por filas (y sin ciclo de repetición de matriz). Es decir:

  $read(A[i,j]) \Longrightarrow$
  Si no estamos en el último ciclo (nº de veces accedido $A[i,j] < P$) :
  • *Prefetching* de $A[i, j+1]$ a $A[i, (j+K)\,\%N]$
  Si estamos en el último ciclo (==P) $\Longrightarrow$ (avance a siguiente fila):
  • *Prefetching* de los K siguientes elementos tanto de la fila actual,
  si quedan, como los primeros de la fila "i+1", si es necesario y $i < M$

- Consideraciones:
  - Dependiendo del tamaño de la cache y el de la matriz, es posible que en cada ciclo todavía quede en la cache parte de la fila, pero se parte del supuesto de que hacer un *prefetching* de algo que está en cache no implica sobrecarga.
  - Si por cada elemento se piden los K siguientes elementos, según se pasa de trabajar con el elemento $A[i,j]$ al elemento $A[i, j+1]$, se piden de forma anticipada los mismos que antes más uno nuevo. Por tanto, sería conveniente llevar a cabo esta operación de una forma agrupada, para evitar pedir uno a uno cada elemento nuevo.

- **REGLA 1 (de otro ejemplo, como la suma de matrices):**
  Si la matriz A fuera un operando de una suma, el *prefetching* realizado debería ser de fila con ventana K, sin ciclo y con avance secuencial por filas (y sin ciclo de repetición de matriz).

- **REGLA 1 (de otro ejemplo diferente):**
  Para especificar la regla correspondiente a otro ejemplo diferente, supongamos que la matriz A toma parte en una operación que implica accederla secuencialmente X veces. En este caso sería conveniente realizar un *prefetching* de fila con ventana K, sin ciclo de repetición por fila, con avance secuencial por filas y con un ciclo de X repeticiones de matriz:

8.4.2. Reglas de *prefetching* sobre la matriz B

El *prefetching* sobre la matriz B es más problemático, ya que, en el caso de la multiplicación de matrices, hay que acceder a dicha matriz por columnas, lo que "rompe" la eficiencia, forzando accesos pequeños. En este caso, cabe plantear diferentes estrategias.

- ***Prefetching* horizontal:**
  Cuando se accede al elemento $B[i, j]$, es conveniente traer una pequeña ventana K' de esa misma fila, correspondientes a los elementos posteriores al pedido, pero siempre con sentido circular (módulo P). Nótese que estos valores no se usarán inmediatamente, sino que hay que esperar hasta que se incremente el índice "j" del bucle, es decir, que se complete el cálculo de un elemento de C. Por tanto, no es posible que K' sea muy grande pues, en caso contrario, la información no cabría en la cache.

- **REGLA 2:**
  Sobre la matriz B, se puede realizar un *prefetching* de fila con ventana circular K' (módulo P), un ciclo de M repeticiones por fila y sin avance por filas (ni ciclo de repetición de matriz), puesto que después de los M ciclos, el algoritmo ya habrá terminado.

- ***Prefetching* vertical:**
  Parece razonable que la solicitud de lectura de $B[i, j]$ cause un *prefetching* de elementos en esa columna: $B[i+1, j], B[i+2, j], \ldots$ Sobre este prefetching "vertical" (de columna) se pueden hacer las siguientes consideraciones:

  - Aunque la lectura adelantada vertical parece razonable en teoría, puede causar lecturas pequeñas obteniendo un mal rendimiento del dispositivo.
  - Por otro lado, si se considera que el *prefetching* es transitivo (véase sección 8.4.3), al dispararse la regla que activa el *prefetching* de $B[i + 1, j]$, lo hará, por transitividad, la regla anterior (REGLA 2), que pedirá la ventana K' en horizontal. Esto mejorará el rendimiento del dispositivo.

  Teniendo en cuenta que la columna puede ser más grande que la cache y que no se desea traerla entera por *prefetching*, se pueden utilizar las siguientes dos reglas combinadas:

  1. El prefetching de fila con ventana K' según la REGLA 2.
  2. La nueva regla correspondiente al *prefetching* vertical (REGLA 3).

- **REGLA 3:**
  Sobre la matriz B, se realizaría un *prefetching* de columna con ventana K", sin ciclo de repetición por columna, con avance secuencial por columnas y con un ciclo de M repeticiones de matriz.

  En el caso de que la columna completa no quepa en la cache, desaparece el efecto positivo de la REGLA 2 ya que lo que se trae anticipadamente en horizontal es expulsado. Por tanto, K' debería ser igual a 0.

8.4.3. *Prefetching* transitivo o inducido

Una regla de *prefetching* permite realizar un *prefetching* de datos "probablemente referenciados" en la próxima lectura. Esta regla de *prefetching* está asociada a un determinado bloque de datos. En el caso de la multiplicación de matrices, está asociada a un elemento o un conjunto de elementos. Si permitimos el uso de reglas de *prefetching* transitivas, el acceso a un determinado bloque de datos, que está asociado a una determinada regla de *prefetching*, permite invocar a su vez otra regla de *prefetching*, asociada al bloque de datos que se ha traido por la primera regla.

No obstante, es necesario controlar esta transitividad, ya que, en determinados casos, puede ocurrir que se acabe leyendo el fichero completo.

8.4.4. *Prefetching* inducido entre ficheros

Este tipo de *prefetching* consiste en "adelantar" datos de un fichero, inducido por accesos en otro fichero.

- **REGLA 4:**
  En el caso de la multiplicación de matrices, se plantea un *prefetching* inducido con ciclo P, de manera que el primer acceso a $A[i, j]$ causa el *prefetching* de $B[j, 0]$, de $B[j, 1]$ y así sucesivamente.

8.4.5. Combinación de reglas de *prefetching*

Juntando todas las reglas planteadas hasta ahora, resulta que un acceso a $A[i, j]$ causa los siguientes accesos:

1. Por la REGLA 1, se hace *prefetching* de los siguientes K elementos de la fila "i" con carácter circular y posibilidad de avanzar a la "i+1" si está en el último ciclo.
2. Por la REGLA 4, se hace *prefetching* de $B[j, x]$ (si es la x-ésima vez que se accede a ese elemento).
3. Por transitividad se aplican las siguientes reglas:
   a) La REGLA 2 y se accede a los K' elementos de la fila j de B.
   b) La REGLA 3 y se accede a los K" elementos de la columna x de B.

8.4.6. Cálculo de *hints* sobre bloques de datos

Los *hints* deben calcularse para un conjunto de datos relacionados. En el caso de aplicaciones de *Data Mining*, los *hints* se calculan por cada tupla o conjunto de tuplas. En el caso de aplicaciones científicas, depende del particionado de los datos. Si consideramos operaciones sobre matrices, almacenadas por filas, los *hints* pueden calcularse por elementos de la matriz o conjuntos de elementos.

En general, se puede almacenar una estructura *hint* por cada conjunto de datos, que denominaremos *bloque*. Este bloque es una división lógica de los datos, por contraposición a bloque físico, el cual se utiliza en los sistemas de ficheros como unidad de transferencia. Los *hints* se dividen en diferentes campos, que dependen de la aplicación de usuario.

8.5. Configuración del sistema MAPFS

Debido a que las aplicaciones de usuario pueden configurar los *hints*, es necesario que el sistema proporcione reglas sintácticas para establecer los parámetros del sistema, los cuales permiten configurar el sistema de E/S.

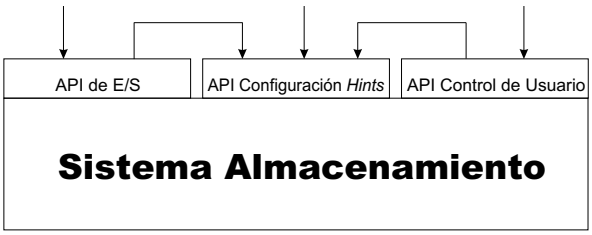


Figura 8.3: API del Sistema de Almacenamiento

Por otro lado, si los *hints* son creados por el subsistema multiagente, también es necesario almacenarlos de una forma predefinida. En cualquier caso, deben introducirse reglas léxicas y sintácticas en el sistema.

El sistema se puede configurar a través de diferentes operaciones, que son independientes de las operaciones de E/S, aunque estas últimas utilizan las primeras. Las operaciones de configuración se dividen en:

1. **Operaciones de configuración de *hints*:** Son operaciones para establecer los *hints* del sistema y, por tanto, permiten modificar y acceder a los diferentes campos de los *hints*.

2. **Operaciones de control de usuario:** Son las operaciones que utilizan de forma directa las aplicaciones de usuario. Las aplicaciones pueden gestionar el rendimiento del sistema sólo a través de este API.

La figura 8.3 muestra los tres niveles en los cuales el API del sistema de almacenamiento se divide, incluyendo el API de las operaciones de E/S.

Como muestra esta última figura, hay tres formas de acceder al módulo de configuración de *hints*:

1. Las operaciones de E/S pueden solicitar información sobre valores de los *hints* y modificarlos.

2. Las operaciones de control de usuario pueden modificar los *hints*. Esta es la forma normal en la que las aplicaciones de usuario interacciona con los *hints*.

3. Con el objetivo de proporcionar flexibilidad al sistema, se puede acceder al módulo de configuración de *hints* directamente a través de las operaciones de configuración de *hints*, es decir, su API. El sistema multiagente puede construir los *hints* a través de esta interfaz, ya que dicho sistema queda integrado dentro del propio sistema de ficheros.

El API de E/S MAPFS se describe en el apéndice A. Las operaciones de configuración de *hints* son las siguientes:

- `Mapfs_Hints * mapHintsNew(int block_ident)`: Esta operación crea una nueva estructura *hint* para el bloque cuyo identificador es `block_ident`.

- `void mapHintsFree(Mapfs_Hints *hint)`: Esta operación libera una estructura *hint*.

- `int mapHintsSet(Mapfs_Hints *hint, int code_field, void * value)`: Esta operación modifica un campo de la estructura *hint* con un valor. La operación devuelve 0 si se realiza de forma correcta y -1 en caso contrario.

- `void * mapHintsGet(Mapfs_Hints *hint, int code_field)`: Esta operación devuelve el valor de un campo concreto de la estructura *hint*. Si el campo no está definido, esta operación devuelve NULL.

Las operaciones de control de usuario tienen una estructura similar:

- `Mapfs_CtrlUser * mapCtrlUserNew(Mapfs_Tuples *tupl)`: Esta operación crea una nueva estructura de control de usuario para un conjunto de tuplas representado por *tupl*.
- `void mapCtrlUserFree(Mapfs_CtrlUser *ctrlUser)`: Esta operación libera una estructura de control de usuario.
- `int mapCtrlUserSet(Mapfs_CtrlUser *ctrlUser, int code_field, void *expr)`: Esta operación modifica un campo de la estructura de control de usuario con una expresión. La operación devuelve 0 si se realiza de forma correcta y -1 en caso contrario.
- `void *mapCtrlUserGet(Mapfs_CtrlUser *ctrlUser, int code_field)`: Esta operación devuelve el valor de un campo concreto de una estructura de usuario de control. Si el campo no es definido, esta operación devuelve NULL.

Estas operaciones hacen llamadas internas a las operaciones del API de configuración de *hints*.

Por otro lado, las expresiones regulares proporcionadas por las aplicaciones de usuario se construyen a través de la siguiente regla, basada en la inducción:

> *exprbool* es una expresión válida;
> si *expr* es una expresión válida, entonces *NOT (expr)* es también una expresión válida;
> si *expr1* y *expr2* son expresiones válidas, entonces *(expr1 AND expr2)* es también una expresión válida;
> si *expr1* y *expr2* son expresiones válidas, entonces *(expr1 OR expr2)* es también una expresion válida;

donde *exprbool* es una expresión booleana aplicada a un atributo o un conjunto de atributos.

Debido al carácter recursivo de la regla, se pueden construir expresiones con infinitos términos. Para que una expresión pueda calcularse en poco tiempo, es necesario limitar esta recursividad. Por tanto, se debe introducir otra regla:

Una expresión debe tener como máximo diez términos.

Este número de términos es configurable por el sistema concreto que utilice MAPFS.

8.5.1. Un ejemplo de aplicación de los modelos de E/S: un sistema de *Data Mining*

Como ejemplo de aplicación se va a considerar un conjunto de tuplas de una tabla con productos y sus precios correspondientes (véase la tabla 8.1)

Podemos utilizar tres expresiones regulares diferentes para optimizar los accesos a la tabla:

- *Barato (B)*: Esta etiqueta se aplica a aquellas tuplas cuyo precio es menor que 10€.
- *Precio Medio (M)*: Esta etiqueta se aplica a aquellas tuplas cuyo precio es mayor que 10€ y menor que 1000€.
- *Caro (C)*: Esta etiqueta se aplica a aquellas tuplas cuyo precio es mayor que 1000€.

Identificador	Nombre	Descripción	Precio
21	"Item21"	"Descripción de Item21"	5€
23	"Item23"	"Descripción de Item23"	6€
26	"Item26"	"Descripción de Item26"	100€
27	"Item27"	"Descripción de Item27"	200€
30	"Item30"	"Descripción de Item30"	1500€
32	"Item32"	"Descripción de Item32"	2000€

Cuadro 8.1: Tabla de productos

Hints	Identificador	Nombre	Descripción	Precio
B	21	"Item21"	"Descripción de Item21"	5€
B	23	"Item23"	"Descripción de Item23"'	6€
M	26	"Item26"	"Descripción de Item26"	100€
M	27	"Item27"	"Descripción de Item27"	200€
C	30	"Item30"	"Descripción de Item30"	1500€
C	32	"Item32"	"Descripción de Item32"	2000€

Cuadro 8.2: Asociación entre la tabla de productos y los *hints*

En este caso, se asocia un *hint* por cada tupla con el valor correspondiente. La tabla 8.2 muestra esta asociación.

En este ejemplo, el bloque corresponde a una tupla. Los *hints* son utilizados en la resolución de consultas relacionadas con el precio y permiten realizar el *caching* y *prefetching* de datos con similares precios, realizando un *clustering* previo de los datos. El uso de los *hints* evita consultar registros que no comparten la misma categoría respecto al precio. La clasificación puede realizarla la aplicación de usuario (a través del API de control de usuario) o el sistema multiagente (directamente a través del API de configuración de *hints*).

8.5.2. Otro ejemplo de aplicación de los modelos de E/S: multiplicación de matrices

Como se describió anteriormente, en el caso de la multiplicación de matrices, es posible aplicar diferentes reglas e incluso una combinación de las mismas. Estas reglas se caracterizan por distintos parámetros, que constituyen las expresiones a partir de las cuales formar la regla sintáctica correspondiente. Estos parámetros y las expresiones asociadas son las que se detallan a continuación:

- Tipo de *prefetching*: Puede ser por filas (horizontal) o por columnas (vertical). Las expresiones correspondientes son: `Tipo = Fila` o `Tipo = Columna`.
- Tipo de ventana: Puede ser circular o no circular. Las expresiones correspondientes son: `Tipo_Ventana = Circular` o `Tipo_Ventana = No_Circular`.
- Tamaño de la ventana: Especifica el número de elementos a leer en cada ciclo de *prefetching*. La expresión asociada es: `Ventana = K`.
- Existencia de ciclo por fila o por columna: Dependiendo del tipo de *prefetching* realizado, establece si se van a llevar a cabo ciclos por fila/columna o no. Las expresiones correspondientes son: `Ciclo_por_fila` o `NOT(Ciclo_por_fila)` y `Ciclo_por_columna` o `NOT(Ciclo_por_columna)`.
- Existencia de ciclo por matriz. Especifica el número de ciclos realizados por matriz. Un valor 0 indica que no existe ciclo por matriz.

- *Prefetching* transitivo. Establece la transitividad de las reglas de *prefetching* utilizadas. Las expresiones asociadas son: `Transitivo` o `NOT(Transitivo)`.

Estas expresiones se combinan mediante operadores `AND` para formar la regla de *prefetching* correspondiente. Dependiendo de la matriz y operación sobre dicha matriz que se considere, estos campos pueden tomar distintos valores, según se describió en la sección 8.4.

Estos campos conforman el factor de *prefetching*, que se describe en la sección 6.3. En los ejemplos descritos, se ha utilizado la capacidad de *hints* junto a la optimización de *caching* y *prefetching*. No obstante, los *hints* se pueden utilizar en conjunción de otras técnicas o algoritmos diferentes, con el objetivo de mejorar las operaciones de E/S. En la sección 10.5 se puede ver un ejemplo de uso de *hints* con un algoritmo que permite descartar líneas de ficheros de entrada a fin de mejorar un algoritmo muy conocido en el campo de las bases de datos, el algoritmo Apriori.

8.5.3. Funciones de evaluación de *hints*

Las expresiones regulares permiten establecer la sintaxis necesaria para configurar los *hints* de un sistema. No obstante, no son lo suficientemente genéricas para ser utilizadas en cualquier tipo de sistemas, ya que pueden existir sistemas que no se adapten adecuadamente a dichas expresiones. Para generalizar la idea anterior, se definen en esta tesis las funciones de evaluación de *hints*, que consisten en funciones del tipo:

```
boolean func_evaluacion(int block_ident, Mapfs_Hints *hints);
```

que reciben un identificador de un bloque de datos que se desea evaluar y la estructura de *hints* utilizada y devuelven un valor verdadero o falso asociado a dicho bloque de disco, dependiendo de los *hints* implantados.

El único requisito que deben cumplir las funciones de evaluación es que estén definidas para todos los identificadores de bloques de datos existentes, asociándoles un valor booleano. Por tanto, son menos restrictivas respecto a la forma de los *hints*. No obstante, su uso de los *hints* es más complejo. Por otro lado, existen dos alternativas respecto a la ubicación de las funciones de evaluación:

- Pueden situarse en la parte servidora. El problema de esta opción es que sobrecarga el servidor de datos. Debido a que MAPFS se caracteriza por descargar al servidor de tareas diferentes a la del acceso a los datos, esta opción no es adecuada.
- Pueden situarse en la parte cliente. De este modo, el cliente proporciona al servidor las funciones de evaluación. El servidor simplemente se encarga de proporcionar los datos y los metadatos (en este caso los propios *hints*). Además tiene sentido el hecho de que sea el cliente el que implemente la función de evaluación, ya que es el que tiene la lógica del procesamiento de la información. Por tanto, esta opción es la seleccionada.

Debido a que el cliente (en nombre de la aplicación de usuario o no) puede generar las funciones de evaluación, que utilizará el servidor para evaluar los diferentes bloques, es necesario establecer una forma de comunicar dichas funciones de evaluación al servidor. A priori, pueden existir diferentes formas de llevar a cabo esta comunicación, a saber:

1. El servidor puede tener una biblioteca de funciones de evaluación, establecidas de forma estática por el cliente. Esta solución no es válida, puesto que no permite incrementar el número de funciones de evaluación.
2. Se utilizan lenguajes interpretados, tipo Java, para permitir que el código sea transmitido y pueda ser ejecutado en cualquier nodo.

3. Se utiliza una biblioteca de funciones de evaluación en el servidor, pero se permite una fase inicial de registro, que permite que el cliente inserte las funciones de evaluación correspondientes. Esta última fase, se puede realizar de diferentes formas: mediante FTP o ya que se está utilizando NFS y MPI, se puede compartir el árbol de directorios.

4. Utilizar movilidad de código. Realizar el proceso anterior, pero permitiendo la migración de código desde el nodo cliente al nodo servidor. Éste detecta cuando no tiene una función de evaluación y se la solicita al cliente en tiempo de ejecución. Esta es la solución más completa.

Ésta es una decisión de diseño, que hay que tener en cuenta a la hora de implementar las optimizaciones de MAPFS.

Por otro lado, debido a que el análisis de los patrones de acceso y el empleo de *hints* es una operación en cierto modo "tangencial" al sistema, es adecuado que se lleve a cabo por parte del sistema multiagente. Por tanto, cuando en esta sección hablamos de cliente, nos referimos a la posibilidad de ejecución en varios nodos clientes por parte de un sistema multiagente, en contraposición a la ejecución en el servidor de los datos[6]. Debido a que el primer prototipo de MAPFS utiliza MPI para la implementación del sistema multiagente, una elección bastante acertada consiste en utilizar también MPI como método de comunicación de las funciones de evaluación.

8.6. Perfiles de E/S

A lo largo de este capítulo y capítulos anteriores se ha dejado constancia de que MAPFS tiene como principal ventaja la flexibilidad, que se manifiesta en la forma de parámetros configurables en diferentes aspectos del propio sistema de ficheros, a saber:

- Configuración de la topología del sistema, a través del establecimiento de los nodos de E/S y sus relaciones. Esta característica se logra a través del uso de los grupos de almacenamiento.
- Especificación de patrones de acceso: Aunque MAPFS consiste en un sistema de E/S de propósito general, puede configurarse para adaptarse a diferentes patrones de acceso de E/S.
- Existen algunas funcionalidades tangenciales al propio sistema de ficheros y que pueden ejecutarse en diferentes nodos del sistema, incluso en los servidores de datos. De hecho, trasladar estas ejecuciones a los servidores de datos ayuda a reducir la latencia y el tráfico de la red. La tecnología de agentes se ha utilizado para implantar estas características.

Todos estos parámetros permiten caracterizar las necesidades de E/S de las aplicaciones que utilizan MAPFS y, por tanto, determinan el perfil de E/S de cada aplicación.

Dentro de esta tesis, se define perfil de E/S de una aplicación a la configuración del sistema MAPFS utilizada para dicha aplicación. Concretamente, un perfil de E/S está compuesto por:

- La topología del sistema de almacenamiento, que consiste en definir el conjunto de retículos principales del sistema.
- Un conjunto de estructuras de control, que permiten configurar y optimizar los accesos de la aplicación al sistema de almacenamiento. Estas estructuras serán traducidas por parte del sistema a *hints*.
- Una descripción del sistema multiagente, que especifica el conjunto multiagente que da soporte al sistema MAPFS.

[6]También es posible que el sistema multiagente se ejecute sobre los servidores de los datos, aunque en este caso se está sobrecargando a dichos nodos con el acceso a los datos y la ejecución de la correspondiente funcionalidad

De una manera formal, se puede definir un perfil de E/S en base a definiciones realizadas en capítulos previos.

Definición 22 *Se define perfil de E/S de una aplicación* `x` *y se denota como* `p(x)` *a la siguiente tupla:*

$$(\{(\sum GP_i, P_G)\}_i, \{(codigo_j, expresion_j)\}_j, \{< \texttt{Id\_Ag}_k, \texttt{G}_l, \texttt{Rol}, \texttt{Red\_Int}_k >\}_{k,l})$$

El perfil de E/S permite definir todos los aspectos anteriormente mencionados. La información recogida por el perfil de E/S es procesada por el sistema de ficheros MAPFS y utilizada para incrementar el rendimiento de E/S de la aplicación correspondiente.

8.7. Resumen

A diferencia de muchos de los trabajos realizados anteriormente, el sistema MAPFS permite que la aplicación defina sus propios "patrones de acceso" a priori, en lugar de caracterizar la carga de trabajo correspondiente. Esto hace que el modelo sea más flexible, sin impedir que se puedan realizar caracterizaciones de la carga y utilizar la misma interfaz para "alimentar" al sistema con esta información.

El análisis y uso de patrones de acceso permite aumentar el rendimiento de las operaciones del sistema de ficheros MAPFS. Este capítulo ha descrito los *hints* como metainformación que permite modelar conocimiento acerca de los datos y patrones de acceso a los mismos. Asimismo, se han analizado dos de las técnicas utilizadas en conjunción con los *hints* para hacer efectivas estas mejoras, a saber, las técnicas de *caching* y *prefetching*. No obstante, como ya se mencionó anteriormente, los *hints* son independientes del algoritmo que se aplique a los mismos, pudiéndose utilizar diferentes técnicas en conjunción con los mismos.

Los perfiles de E/S constituyen la especificación de los parámetros de configuración de MAPFS apropiados para cada una de las aplicaciones que hacen uso del sistema MAPFS. Estos perfiles recogen información sobre la topología del sistema, las estructuras de control de usuario y una descripción del MAS que da soporte al sistema de ficheros.

Capítulo 9

Implementación de MAPFS

9.1. Introducción

En los capítulos anteriores se ha analizado el sistema MAPFS desde un punto de vista descriptivo, incidiendo en aquellos aspectos que le caracterizan y diferencian del resto de sistemas de ficheros que engrosan la bibliografía sobre E/S paralela. No obstante, a fin de que este trabajo sea exhaustivo, es importante ofrecer pautas sobre la implementación e implantación del sistema MAPFS. Este capítulo muestra los aspectos más importantes de la implementación del sistema, relacionándolos con las características del sistema descritas en capítulos previos.

9.2. Arquitectura global del sistema MAPFS

Esta primera sección muestra el sistema MAPFS desde un punto de vista arquitectónico, describiendo las características que afectan al sistema de forma general. La implementación de los dos subsistemas que componen el sistema se describen en apartados posteriores.

En primer lugar es importante destacar que el sistema MAPFS ha sido diseñado para su uso en clusters de estaciones de trabajo. Como ya se ha mencionado anteriormente, en la computación paralela y distribuida ha habido una tendencia incremental hacia el uso de clusters, debido principalmente a su bajo coste y su facilidad de integración. Por otro lado, existen dos características de los clusters que son especialmente útiles en el caso de MAPFS, a saber:

- Un cluster es un sistema que permite procesamiento paralelo o distribuido. Es la mezcla de ambas características la que permite que un cluster sea la elección hardware más acertada en el caso de implementar MAPFS, debido a que este sistema de ficheros también hace uso de dichas características.

- Un cluster tiene la capacidad de establecer un sistema RAID por software, debido a que proporciona la capacidad de tener datos redundantes, incrementando de este modo la disponibilidad de la información almacenada en el sistema. De hecho, para implementar un sistema de ficheros

paralelo, se pueden utilizar los discos asociados a cada estación de trabajo, empleando una de las particiones para proporcionar tolerancia. Esta solución contrasta con la empleada por sistemas que permiten realizar RAID por hardware, más sofisticados pero que resultan excesivamente caros.

Otro de los objetivos del sistema MAPFS es el empleo de tecnologías de servidor ya consolidadas, utilizando dichos servidores como meros repositorios de datos y proporcionando características adicionales a partir de los módulos implementados en la parte cliente.

Como primer prototipo, se ha utilizado el sistema de ficheros NFS en la parte del servidor. Como ya se destacó, NFS es un sistema de ficheros en red, desarrollado inicialmente por Sun, que permite el acceso a ficheros remotos. El uso de servidores NFS presenta varias ventajas frente a otras propuestas, a saber:

1. Se integra muy bien en los sistemas distribuidos convencionales, ya que prácticamente todos ellos disponen de servidores NFS. Además, el empleo de particiones distribuidas puede convivir con el empleo de particiones tradicionales.

2. El sistema de ficheros NFS se adapta fácilmente a un cluster de estaciones de trabajo.

3. Se facilita la construcción del sistema de ficheros paralelo, puesto que se parte de un servidor existente, cuyo funcionamiento se encuentra ampliamente probado. Este enfoque es diferente a lo que hacen todos los sistemas de ficheros paralelos actuales, que se basan en construir desde cero tanto el cliente como el servidor del sistema de ficheros paralelo. Esto hace más difícil su integración en entornos distribuidos, como lo demuestra la experiencia del grupo de investigación en el desarrollo del sistema de ficheros paralelo ParFiSys [CPdM+96], [GCPdM99].

4. Permite acceder en paralelo tanto a los datos de diferentes ficheros, como a los de un mismo fichero, lo que reduce el cuello de botella que plantea el acceso a servidores convencionales.

5. Mejora el uso de los recursos del sistema ya que la distribución de los datos entre diferentes servidores NFS conduce a un mejor equilibrio de la carga, ya que evita que haya servidores muy ocupados mientras otros tienen mucho espacio libre.

6. Se adapta a la naturaleza heterogénea de un sistema distribuido, ya que permite utilizar de forma conjunta servidores residentes en máquinas y sistemas operativos de diferentes fabricantes. Así por ejemplo, se podría construir una partición distribuida utilizando máquinas con sistemas operativos Linux, UNIX y Windows 2000 sin necesidad de realizar cambios en los clientes del sistema de ficheros.

Todas las implementaciones de NFS soportan el protocolo NFS, un conjunto de llamadas a procedimientos remotos (RPC) que permiten a los clientes realizar operaciones sobre ficheros remotos. En estas características nos hemos basado para implementar este primer prototipo, haciendo uso de estas llamadas RPC para acceder a los ficheros almacenados en los servidores NFS. El protocolo NFS es independiente del sistema operativo empleado y es el sistema de ficheros en red más ampliamente utilizado. Aunque originariamente se desarrolló para su uso en redes de sistemas UNIX, se ha extendido a otros sistemas operativos como Windows 2000. De este modo, el uso de NFS también permite beneficiarse de la heterogeneidad del entorno. Por tanto, esta primera fase plantea la construcción de un sistema de ficheros paralelo que emplee particiones distribuidas a través de varios servidores NFS. No obstante, de cara a futuras implementaciones, se propone un sistema que permita acceso a servidores tradicionales o distribuidos ya implementados, mediante el uso de una interfaz unificada, que debe ser traducida a las llamadas concretas que permiten comunicar la parte cliente con la parte servidora.

De esta forma, incluso se podrían utilizar diferentes tipos de servidores como repositorios de datos, proporcionando heterogeneidad software en la parte del servidor. El uso de clusters heterogéneos dota a su vez al sistema de heterogeneidad hardware.

A continuación se va a detallar la arquitectura del sistema de ficheros MAPFS, mostrando tanto la parte cliente como servidora.

9.2.1. Arquitectura en tres niveles de MAPFS

El problema de la arquitectura de MAPFS surge de la necesidad de mantener los servidores de ficheros inalterados. Por tanto, toda la funcionalidad adicional debe llevarse a cabo por los clientes MAPFS. No obstante, el papel de cliente tiene como característica inherente su escasa capacidad de procesamiento y el hecho de que sobrecargar el cliente con toda la funcionalidad, convierte a este elemento en cuello de botella del sistema. Esto rompe con los requisitos de alto rendimiento del sistema MAPFS. La solución propuesta para resolver esta disyuntiva es el uso de una arquitectura de 3 niveles, cuyas capas se describen a continuación:

- **Clientes MAPFS:** Los clientes toman un papel destacado en el sistema MAPFS. Los clientes corresponde al subsistema de ficheros (véase sección 5.4), que contiene la implementación de las operaciones de E/S. Estas operaciones se traducen finalmente en llamadas de acceso a los datos de los servidores.
- **Servidores finales de almacenamiento:** A fin de cumplir los requisitos de diseño del sistema MAPFS, los servidores utilizados en el sistema no deben modificarse. Estos servidores almacenan la información del sistema y deben ser accesibles a través de una interfaz utilizada por la parte cliente.
- **Sistema multiagente de apoyo a los clientes MAPFS:** Con el objetivo de descargar a los clientes de todo el procesamiento, se utiliza un sistema multiagente, que puede ejecutarse en diferentes nodos, incluidos los nodos servidores de datos. Esto no contradice el requisito de no modificar los servidores, ya que este proceso es totalmente independiente y queda desacoplado de la propia funcionalidad del servidor de ficheros. El sistema multiagente permite llevar a cabo cinco tareas fundamentalmente:
  1. Soporte al sistema de ficheros, interactuando con el subsistema de ficheros en la tarea de recuperación de los datos.
  2. Gestión de la cache del sistema.
  3. Gestión de la tolerancia a fallos.
  4. Gestión de los grupos de almacenamiento.
  5. Creación y gestión de los *hints*.

La figura 9.1 muestra la división en 3 capas del sistema y las relaciones entre ellas, así como las interfaces entre las mismas.

En la jerarquía más alta de la figura aparece la interfaz del subsistema de ficheros, que está formada por las llamadas al propio sistema de ficheros. Este subsistema se comunica con dos módulos:

1. Con el subsistema multiagente, a través de la interfaz MAPFS_MAS. Este subsistema sirve de apoyo al propio subsistema de ficheros.
2. Con los servidores de datos, a través del uso de RPC o alguna tecnología de acceso a los mismos. En el caso de uso de servidores NFS, se utiliza la interfaz RPC a los servidores NFS, especificada en [NFS95].

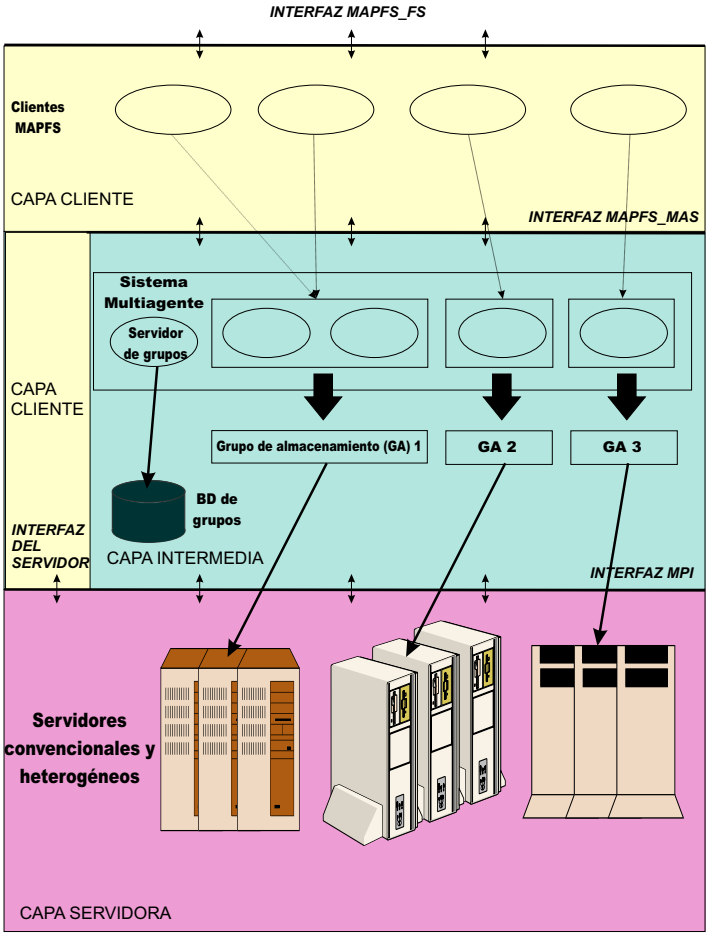


Figura 9.1: División en 3 capas o niveles de la arquitectura de MAPFS

Respecto al subsistema multiagente, lo primero que destaca en la figura es el uso de MPI como tecnología utilizada para la implementación del marco en el que se sitúa el propio subsistema. Como ya se ha descrito en el capítulo 6, el uso de arquitecturas de agentes no es adecuada en este entorno, debido principalmente a dos motivos:

1. La programación de sistemas interacciona a muy bajo nivel con el propio sistema. El paradigma de agentes permite una interacción a un nivel mucho más alto.

2. La eficiencia es un requisito muy estricto en el caso de la programación de sistemas. Introducir una capa de abstracción siempre implica una pérdida de eficiencia en la ejecución de cualquier sistema.

Por tanto, se ha eligido como tecnología de implementación MPI, principalmente por las siguientes características:

1. Constituye una interfaz estándar de paso de mensajes, que permite a diferentes agentes del sistema comunicarse a través del envío de mensajes.

2. El paso de mensajes también permite llevar a cabo la sincronización de los procesos.

3. MPI se utiliza ampliamente en los clusters de estaciones de trabajos.

4. El uso de MPI permite incrementar la eficiencia de la solución, debido a su capacidad para la programación de aplicaciones paralelas.
5. Permite gestión dinámica de procesos.
6. Da soporte a tipos de datos derivados.
7. Proporciona operaciones que permiten establecer diferentes topologías de comunicación o modificar el entorno de los procesos.

Estas propiedades de MPI se utilizan para proporcionar las características propias de los agentes:

- **Autonomía:** La capacidad que tiene MPI de crear de forma dinámica procesos permite el uso de procesos independientes que se comunican entre sí y que se caracterizan por su autonomía frente al proceso global.
- **Reactividad:** La reacción al entorno o a la ejecución de otros procesos llega a un determinado proceso a través de un mensaje MPI. De hecho, las *performatives* KQML empleadas en la comunicación entre los agentes son traducidas a mensajes MPI en la implementación del sistema MAPFS, como se describe en la sección 9.4.
- **Movilidad:** Permitir que un sistema multiagente tenga la característica de movilidad, requiere del uso de arquitecturas que permiten la migración de procesos o funciones similares, tales como MOSIX [BO98]. No obstante, estas arquitecturas son más complejas. Por otro lado, el sistema de ficheros MAPFS no necesita una movilidad absoluta. Realmente lo que se ha implementado en el caso de MAPFS es lo que hemos denominado "movilidad geográfica de funcionalidad", por contraposición a la movilidad de proceso y que implica que se puede decidir en qué nodo ejecutar una determinada funcionalidad, pero no recrear un proceso en otro nodo. De este modo, no es necesario guardar el estado de los procesos que migran, sino que son las funcionalidades las que cambian de lugar de ejecución, deteniéndolas y volviéndolas a lanzar.

Como implementación de MPI, se ha utilizado LAM/MPI [LAMb], que incluye muchas de las características de MPI-2, útiles para nuestros propósitos, entre las que destaca la gestión dinámica de procesos. Además esta implementación proporciona tolerancia a fallos y es de libre distribución.

9.3. Implementación del subsistema de ficheros

Debido a que el sistema de ficheros MAPFS se beneficia principalmente de las características de paralelismo y distribución, es necesario elegir una forma de llevar a cabo el procesamiento paralelo y distribuido. Respecto a la primera característica, el procesamiento paralelo se logra a través del uso de múltiples *procesos* que procesan las peticiones de E/S de una forma paralela. La ejecución de los procesos se realiza a través del uso de MPI. Respecto a la segunda característica y debido a que el sistema de ficheros que se utiliza inicialmente es NFS y que dicho sistema utiliza como método de comunicación las RPC, el sistema de ficheros MAPFS también va a utilizar este mecanismo para acceder a los ficheros que se almacenan de una forma distribuida. En la figura 9.2 queda representada la arquitectura del subsistema de ficheros de MAPFS, MAPFS_FS, en el que se muestra el acceso a los nodos distribuidos.

Por tanto, el subsistema se puede dividir en tres capas o niveles:

- **Capa de comunicación**: Este nivel se encarga de realizar las operaciones que permiten la comunicación con los servidores, de forma que se pueda llevar a cabo el acceso a los ficheros de forma remota. Esta capa utiliza el mecanismo de comunicación de RPC.

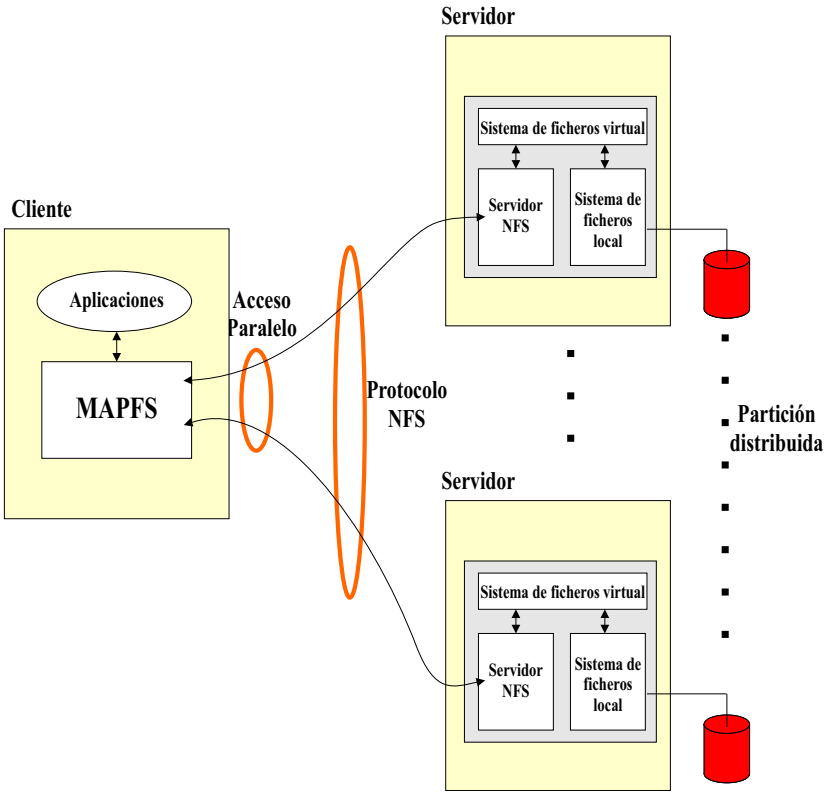


Figura 9.2: Arquitectura del subsistema MAPFS_FS

- **Capa de paralelismo** Esta capa es la que interacciona tanto con el subsistema de ficheros como con el subsistema multiagente. Se utiliza MPI como interfaz de paso de mensajes. MPI permite el uso de ficheros de esquema, que especifican los nodos en los cuales se van a ejecutar las copias del cliente MAPFS. Estos ficheros permiten modificar la topología del sistema de forma dinámica durante la ejecución de las aplicaciones. En la sección 9.4 se muestra cómo se utilizan dichos ficheros de esquema.
- **Interfaz del sistema de ficheros** Esta interfaz se ha descrito en la sección 5.4 y constituye el medio de interacción con las aplicaciones de usuario.

En la figura 9.3 se muestra la estructura del subsistema de ficheros así como las llamadas que permiten invocar a los diferentes componentes del mismo.

9.4. Implementación del subsistema multiagente

Como hemos afirmado anteriormente, se utiliza un subsistema multiagente como soporte a la recuperación de datos, proporcionando dicho subsistema características que permiten incrementar el rendimiento de las operaciones de E/S realizadas.

Cada cliente MAPFS está asociado a un único grupo de almacenamiento y este grupo de almacenamiento asociado a un determinado sistema multiagente. La topología utilizada para la ejecución del cliente MAPFS constituye la partición de nodos correspondientes a dicho grupo de almacenamiento.

En la figura 9.4 se muestra el flujo de control en la jerarquía de procesos agentes de un determinado grupo de almacenamiento. La base de datos de grupos se describe en la sección 9.5.1.

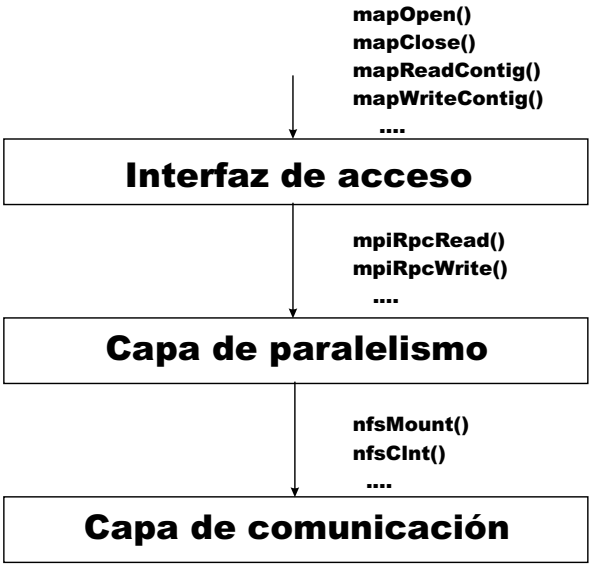


Figura 9.3: Estructura en niveles del subsistema MAPFS_FS

En la base de datos de grupos se encuentra almacenada la información relacionada con la topología de nodos para un determinado fichero. La ejecución de los nodos extractores sobre dicha topología se realiza mediante el uso de la llamada `MPI_Comm_spawn()` y los ficheros de esquema, los cuales utilizan la estructura `MPI_Info`.

La rutina `MPI_Comm_spawn()` permite arrancar un determinado número de procesos MPI y establecer comunicación con ellos, a través de un intercomunicador. Los procesos arrancados son referenciados como procesos hijos. Uno de los parámetros de esta llamada es una estructura de tipo `MPI_Info`, que contiene información adicional sobre la creación de los procesos. Esta estructura consiste en un conjunto de pares `(clave,valor)`. Una de las claves interpretadas por la llamada `MPI_Comm_spawn()` es `SCHEMA_NAME`, que permite especificar un nombre de fichero esquema, que contiene la topología de servidores donde se va a ejecutar los procesos hijos.

En el caso de MAPFS, el agente distribuidor se encarga de construir de forma dinámica el fichero esquema, según la información almacenada en la base de datos de grupos, y arrancar las copias del proceso extractor en la topología especificada por dicho fichero.

Cada agente extractor ejecuta a su vez una copia del agente cache, según la misma dinámica. En la sección 6.3 se describe la problemática relativa a este tipo de agentes. Respecto a su implementación, MPI proporciona una utilidad para llevar a cabo el proceso de *caching*. Esta utilidad permite adjuntar información, denominada atributos en terminología MPI, a los comunicadores que se utilizan para la comunicación. Mediante esta utilidad es posible:

- Pasar información entre diferentes llamadas, asociándola con un comunicador MPI.
- Recuperar rápidamente la información.
- Garantizar que nunca se recupera información obsoleta, incluso si el comunicador se libera y se utiliza posteriormente.

Por último, la tolerancia a fallos es una característica opcional, tal y como se ha definido en el sistema. En caso de aplicarse, cada agente extractor ejecuta una copia del agente de tolerancia a fallos.

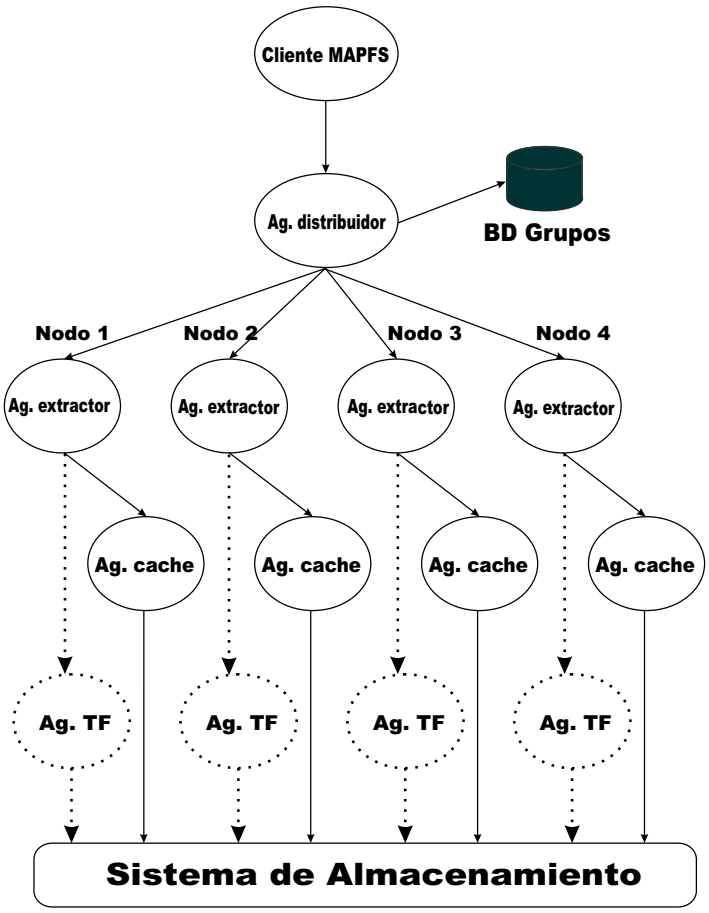


Figura 9.4: Flujo de control en el subsistema multiagente correspondiente a un determinado cliente MAPFS

El problema de la tolerancia a fallos también se puede resolver a nivel de implementación mediante el uso de MPI. MPI permite proporcionar tolerancia a fallos mediante un esquema cliente/servidor, siguiendo la estrategia que se enuncia a continuación:

- Crear un proceso hijo por cada uno de los procesos.
- Establecer un manejador de errores para la comunicación entre cada par (padre, hijo). El error que se puede generar es *MPI_ERRORS_RETURN*.
- Si la comunicación padre-hijo tiene un error, asumir que el proceso que no responde está muerto y liberar el intercomunicador correspondiente.
- Crear otro proceso hijo para reemplazar el proceso muerto. Esto es opcional, pero conveniente para seguir proporcionando el servicio de tolerancia a fallos.

En la figura 9.5 se muestra gráficamente la implementación de esta característica.

Finalmente, uno de los aspectos más relevantes de un sistema multiagente es precisamente la comunicación. Como se describe en el capítulo 6, a fin de comunicar los diferentes agentes, se utilizan *performatives* KQML.

KQML permite definir una abstracción del nivel de transporte entre diferentes agentes. Este nivel debe cumplir los siguientes requisitos:

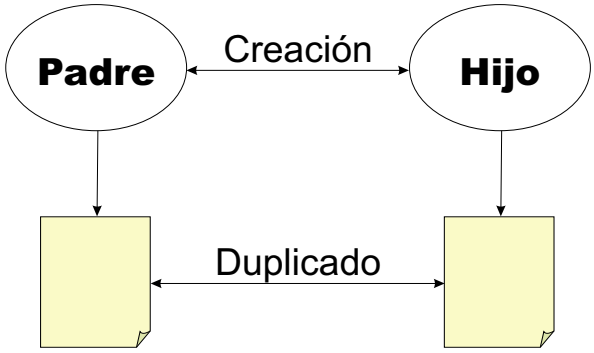


Figura 9.5: Implementación de la tolerancia a fallos en el sistema MAPFS

- Los agentes están conectados mediante enlaces de comunicación unidireccionales.
- Estos enlaces pueden tener un retardo de transporte distinto de cero.
- Cuando un agente recibe un mensaje, conoce el enlace por el cuál llega el mensaje.
- Cuando un agente envía un mensaje, puede dirigirlo a un determinado enlace.
- Los mensajes enviados a un único destino, llegan en el orden en el que se enviaron.
- El envío de mensajes es fiable.

Esta abstracción se puede implementar de múltiples formas. Los enlaces pueden implementarse como conexiones TCP/IP sobre Internet, existentes durante la transmisión de los mensajes. También pueden implementarse mediante mecanismos de comunicación entre procesos, tales como colas de mensajes o sockets. Sin tener en cuenta cómo se lleve a cabo la comunicación realmente, KQML considera la comunicación entre los agentes como paso de mensajes punto a punto. MPI es una buena opción para la implementación, por las características descritas anteriormente y porque se adapta completamente a los requisitos demandados por este tipo de *performatives*.

Debido a que la implementación final se lleva a cabo mediante MPI, es importante conocer cómo se traducen en el sistema las *performatives* a mensajes MPI. La tabla 9.1 muestra las traducciones de algunos de las *performatives* más utilizadas.

Hay que destacar que MPI sólo resuelve el problema de comunicación. La semántica es proporcionada por el contenido de los mensajes, a través de las estructuras MPI y por el procesamiento del mensaje por parte del agente receptor.

9.5. Implementación de los grupos de almacenamiento

Esta sección muestra los aspectos más relevantes sobre la implementación de los grupos de almacenamiento en el sistema de ficheros MAPFS.

La información sobre los grupos de almacenamiento debe ser conocida por el cliente MAPFS, ya que los servidores se utilizan como meros contenedores de información. Para representar los grupos de almacenamiento principales, el cliente MAPFS utiliza la matriz de agrupación principal, que necesita muy poco espacio para su almacenamiento. No obstante, en el funcionamiento normal del sistema, es necesario también utilizar los grupos de almacenamiento secundarios y de replicación. Para homogeneizar la información correspondiente a todos los tipos de grupos de almacenamiento se utiliza una base de datos, que se describe a continuación.

Performative KQML	Mensajes MPI
ask-one	MPI_Send MPI_Recv de un único elemento básico
ask-all	MPI_Send MPI_Recv de un vector de elementos
stream-all	MPI_Send MPI_Recv por cada uno de los elementos a enviar + uno para eos
eos	MPI_Send
tell	MPI_Send
untell	MPI_Send
deny	MPI_Send
insert	MPI_Send
uninsert	MPI_Send
delete-one	MPI_Send
delete-all	MPI_Send
undelete	MPI_Send
achieve	MPI_Send
unachieve	MPI_Send
error	MPI_Send
sorry	MPI_Send
forward	MPI_Send Interpretación de la *p*erformative que incluye

Cuadro 9.1: Traducción de *p*erformatives KQML a mensajes MPI

9.5.1. Base de datos de grupos

A fin de gestionar de una forma eficiente los grupos de almacenamiento, se utiliza una base de datos que contiene la configuración en un instante determinado del sistema respecto a su división en grupos. Dicha base de datos tiene tres tablas diferentes que describen el estado de agrupación del sistema:

1. Una tabla que almacena información general sobre cada grupo de almacenamiento del sistema. Hay una entrada (o tupla) por cada grupo de almacenamiento del sistema. De cara a implementar un grupo de almacenamiento, es necesario identificarlo de forma única en el sistema. Para designar un grupo de almacenamiento se utiliza un identificador de tipo *Mapfs_Group*.

   Además de este campo, que constituye la clave, por cada entrada se debe almacenar la siguiente información:

   - El tipo de grupo de almacenamiento: Es necesario distinguir si se trata de un grupo de almacenamiento principal, secundario o de replicación.
   - La jerarquía de cada grupo de almacenamiento. En el caso de grupos secundarios, hay qué especificar el grupo principal al qué pertenece. En el caso de grupos de replicación, es necesario indicar los grupos proveedores. Por último, en caso de grupos principales, este campo no tiene ningún significado.
   - El espacio libre del grupo de almacenamiento: para calcular este parámetro, se debe tener en cuenta el espacio libre de cada uno de los nodos de almacenamiento que forman el grupo. Se puede utilizar una función sobre este conjunto de datos para obtener el parámetro requerido. No obstante, este cálculo depende de la función de distribución utilizada. Si se utiliza la

función *round-robin*, debido a que la información se distribuye en forma de rodajas en cada uno de los nodos de un grupo, el espacio disponible viene marcado por el espacio libre del nodo con menos capacidad de almacenamiento, por lo que la función *mínimo* parece una buena opción, aunque una función más sofisticada debería tener en cuenta que no siempre se realiza la distribución de forma equitativa (los nodos más próximos al nodo inicial de un fichero tendrán mayor o igual cantidad de datos de dicho fichero que los nodos más lejanos).

- Parámetros de agrupación: se trata de campos calculados para cada uno de los grupos y que pueden variar a lo largo del tiempo, debido a su carácter dinámico (véase sección 7.6).

2. Una tabla que relacione los grupos de almacenamiento con los servidores que pertenecen a los mismos. En el caso de los grupos principales, bastaría con utilizar la matriz de agrupación. Por homogeneidad, esta información se muestra como tabla. Además, esta tabla contiene un número de orden para secuenciar los servidores o nodos correspondientes a un determinado grupo de almacenamiento.

3. Una tabla que relacione ficheros con grupos de almacenamiento. A partir de esta tabla y la anterior, podemos conocer la distribución de un determinado fichero en un grupo de almacenamiento.

9.6. Implementación de las estructuras de control de usuario y *hints*

Las estructuras de control de usuario permiten, a través de la interfaz descrita en la sección 8.5, establecer información de control que incremente el rendimiento de las operaciones de E/S. Esta información es traducida a *hints* por parte del sistema y almacenada, procediéndose a su uso en dichas operaciones.

En el capítulo 5 se detalla la estructura *map-node* y se describen los *hints* como uno de los campos de dicha estructura. Esto implica que los *hints* son creados y utilizados por fichero. Esto ofrece flexibilidad a las aplicaciones, ya que se pueden especificar diferentes *hints* para diferentes ficheros. No obstante, almacenar los *hints* en el *map-node* supone un problema, debido a que esta estructura no permite que aplicaciones diferentes puedan utilizar diferentes *hints* para un mismo fichero. Este problema se puede resolver asociando una estructura *hint* a un descriptor de fichero abierto. Por este motivo, la estructura *map-node* pierde el campo *hints*, quedando como en la figura 9.6.

De este modo, los *hints* pasan a ser metadatos asociados a ficheros abiertos y sólo se pueden utilizar durante el tiempo que ese fichero esté abierto, perdiéndose la información correspondiente al cerrarse. De ahí, que los *hints* puedan almacenarse en memoria por parte del sistema de ficheros. La estructura *hint* consiste en una lista de pares `(clave, valor)`.

La operación que permite aplicar una estructura *hint* a un determinado fichero MAPFS abierto previamente es:

```
void mapHintsApply( Mapfs_File file, Mapfs_hints *hint);
```

Por su parte, la operación que debe utilizar una aplicación para aplicar una determinada estructura de control a un determinado fichero MAPFS abierto previamente es:

```
void mapCtrlUserApply( Mapfs_File file, Mapfs_CtrlUser *ctrlUser);
```

9.7. Interfaz gráfica del sistema MAPFS

Con el objetivo de evaluar el sistema y proporcionar aplicaciones útiles para el usuario del sistema MAPFS, se han implementado un conjunto de herramientas que permiten convertir ficheros tradicio-

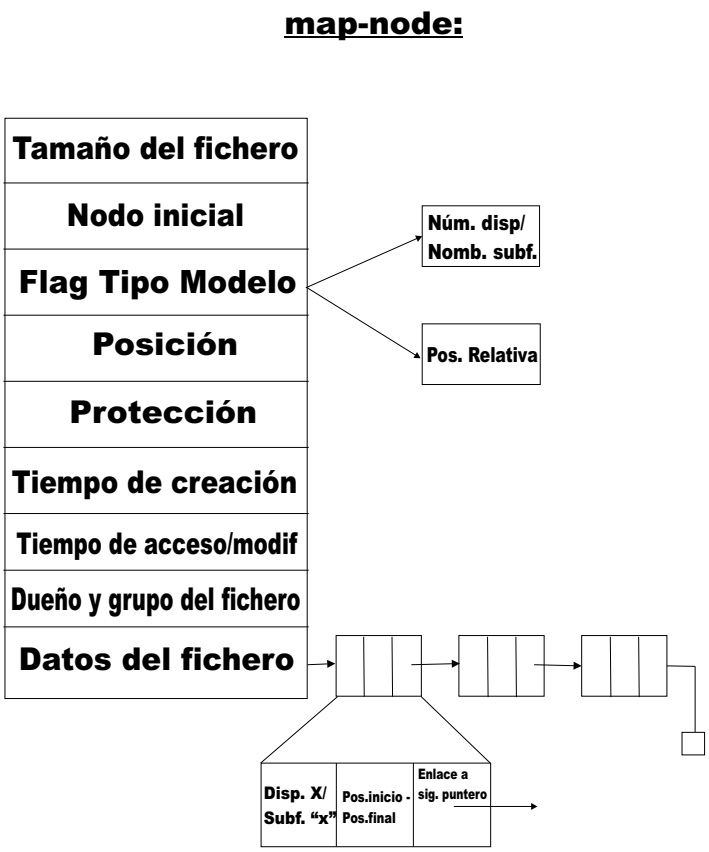


Figura 9.6: Estructura del *m*ap-node, eliminando el campo de *h*ints

nales en ficheros MAPFS y viceversa, así como otras funciones interesantes (véase sección 11.5). Estas utilidades invocan internamente a la interfaz del sistema de ficheros MAPFS.

Existen dos formas de ejecutar estas herramientas MAPFS, a saber, en modo mandato y en modo gráfico. En modo mandato, implica arrancar un cliente MPI que se encargue de ejecutar la herramienta correspondiente. Para facilitar esta labor, se ha implementado una interfaz gráfica en Java, denominada `MapView`, que permite realizar las operaciones de una forma sencilla e intuitiva.

Esta interfaz gráfica se puede utilizar de dos modos diferentes, *modo operacional* y *modo visualización*.

El *modo operacional* permite ejecutar las utilidades, interaccionando directamente con los ficheros. La figura 9.7 muestra el *modo operacional* de la interfaz gráfica del sistema.

Por otro lado, el *modo visualización* permite visualizar la topología del sistema, mostrando los grupos de almacenamiento y dando la posibilidad al usuario de interaccionar con el sistema, modificando dicha topología. Entre las operaciones que se pueden realizar se encuentran las siguientes:

1. Creación de nuevos grupos de almacenamiento.
2. Modificación de la relación de agrupación.
3. Cambio de un servidor de grupo.
4. Eliminación de un grupo de almacenamiento.
5. Comprobación de que un servidor está operativo (*tping*).

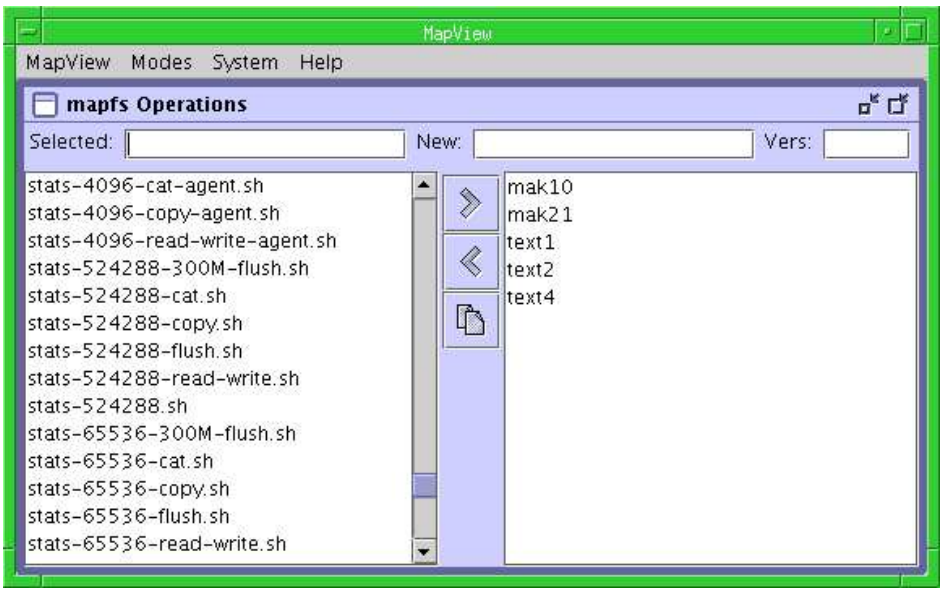


Figura 9.7: *M*odo operacional de la interfaz gráfica del sistema MAPFS

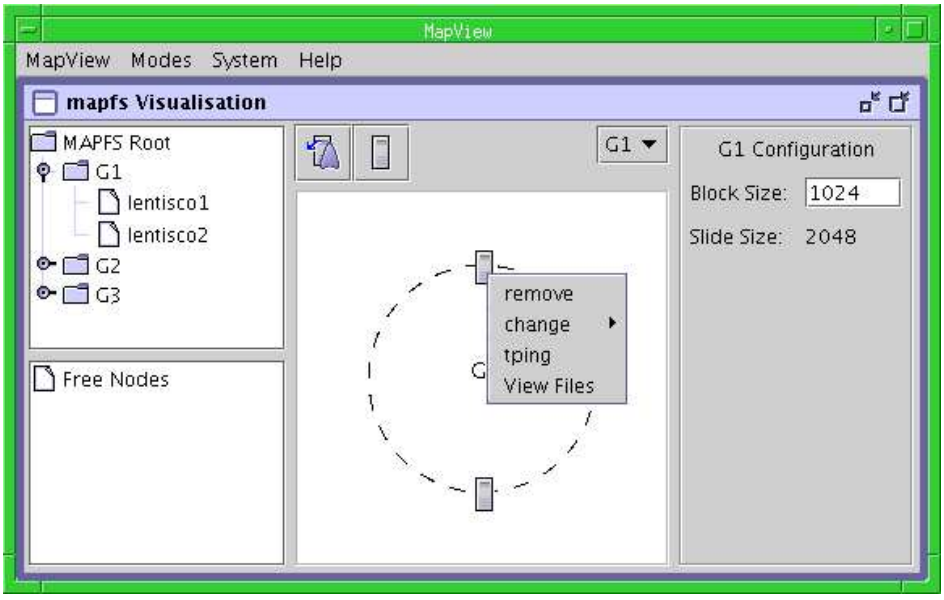


Figura 9.8: *M*odo visualización de la interfaz gráfica del sistema MAPFS

6. Visualización de los ficheros almacenados en un determinado grupo de almacenamiento.

La figura 9.8 muestra el *modo visualización* de la interfaz gráfica del sistema.

9.8. Resumen

Este capítulo ha descrito algunos aspectos de implementación, necesarios para comprender de forma completa la arquitectura del sistema MAPFS y encajar las distintas piezas que componen la misma. Concretamente, el capítulo incluye características de la implementación de los subsistemas en

que se divide el sistema global, de los grupos de almacenamiento y de las estructuras de control y *hints*. La última sección describe la interfaz gráfica proporcionada para facilitar el uso de las herramientas relacionadas con el sistema MAPFS.

Capítulo 10

Posibles aplicaciones de MAPFS

10.1. Introducción

Un sistema de ficheros sirve de soporte a aplicaciones de niveles superiores y tiene sentido siempre que sea capaz de proporcionar su servicio de una forma eficiente y lo más transparente posible. Es por ello que este trabajo, además de presentar el sistema de ficheros MAPFS, también analiza y evalúa las posibles aplicaciones que se puedan desarrollar sobre el mismo.

Las aplicaciones precisan cada vez más potencia y capacidad de almacenamiento. Tradicionalmente estas aplicaciones con una mayor necesidad de potencia de cómputo han procedido del campo de la investigación científica e ingenieril, donde es necesario realizar simulaciones de sistemas complejos, llevándose a cabo una gran cantidad de calculos numéricos. Hoy en día, también las aplicaciones comerciales necesitan procesar una gran cantidad de datos. Como ejemplos de esta tendencia existen aplicaciones que permiten reconocer la voz o la imagen, o aplicaciones de *Data Mining* que permiten reconocer patrones a través de una cantidad ingente de datos. Otras aplicaciones encajan dentro de los campos de la telemedicina, de la bioinformática o en el campo de los juegos (entornos virtuales, por ejemplo).

HPDA (*High Performance Data Analysis*) es un sistema que surge como propuesta para dar solución a estas necesidades y, por tanto, permite llevar a cabo un análisis de grandes cantidades de datos, logrando un alto rendimiento. HPDA utiliza MAPFS como sistema de ficheros.

Aunque HPDA surge como un entorno independiente de aplicación, dicho sistema va a ser utilizado básicamente por dos tipos de aplicaciones: aplicaciones de *Web Mining* y aplicaciones de bioinformática. Estos campos son lo suficientemente versátiles y representativos como para validar el sistema integral. Además, ya en [MZ00] se enuncia como propuesta el uso de sistemas de ficheros paralelos para la construcción de sistemas escalables paralelos de *Data Mining* (PKDD).

10.2. *Web Mining*

El análisis de las enormes cantidades de datos que las organizaciones generan día a día como consecuencia de las operaciones realizadas, dio lugar a la aparición a finales de los 80 del término KDD para describir el conjunto de acciones encaminadas a analizar dichos datos para obtener información de interés. *Data Mining*, siendo tan sólo una parte de tal proceso, es la fase que más atención ha atraído y, como consecuencia, en los diez últimos años son numerosos los sistemas desarrollados tanto en universidades como para su uso comercial (*Intelligent Miner*, *Quest*, *DBMiner* o *Clementine* por nombrar algunos).

No obstante Internet supone un nuevo reto para este tipo de técnicas y sistemas. Los datos recogidos vía web difieren en naturaleza y contenido a los datos que tradicionalmente se estaban analizado. Por otra parte, el análisis de usuarios que navegan por la Web requiere una primera identificación de los mismos, cuestión que en la actualidad es imposible realizar debido a: (i) infraestructura subyacente: el protocolo *http* es no persistente y (ii) las herramientas y sistemas tradicionales no están ajustadas para este tipo de información. La obtención de un sistema capaz de analizar perfiles de navegantes web en tiempo real es de vital importancia para todo tipo de organizaciones que han comenzado a desarrollar su actividad en Internet en los últimos tiempos y que como resultado de ello están perdiendo el trato personalizado con sus usuarios.

La combinación de las dos áreas, *Data Mining* y la web da lugar a lo que se conoce como *Web Mining*, que se define como "la aplicación del proceso de KDD a datos recogidos en los *logs* de los servidores, o la información recogida en la web, ... Así, se puede decir que *Web Mining* consiste en hacer uso de las técnicas de *Data Mining* para automatizar el descubrimiento y extracción de información a partir de documentos y servicios existentes en la web" [Etz96].

Es importante destacar que los sistemas que se están desarrollando asumen tratamiento de datos desde ficheros planos no teniendo una infraestructura de bases de datos adecuada para los datos que se están recogiendo.

Internet supone un flujo de datos nuevo y desconocido. Este flujo de datos "clickstream" contiene cada referencia de cada uno de los movimientos realizados por un usuario mientras estaba conectado en un sitio Web. Como consecuencia, este flujo de datos es un registro del comportamiento de los usuarios mucho mejor que cualquiera de los registros de datos existentes hasta el momento. El flujo de datos generado en cualquier sitio Web es una serie de acciones microscópicas que se pueden unir en sesiones y que analizados de manera eficiente permitiría entender y descubrir muchos de los comportamientos de los usuarios. Analizando convenientemente este flujo de datos es posible obtener perfiles de navegantes, predecir futuros comportamientos y personalizar páginas web dinámicamente entre otras.

No obstante, la información que se recoge en los servidores web y en el los servidores de aplicaciones no es completa, y no se está analizando correctamente pues básicamente se están aplicando heurísticas y herramientas de visualización y OLAP, lo que hace que no sea útil para la mejora de las operaciones llevadas a cabo en la Web como pueden ser el comercio electrónico en todas sus variantes (B2B, B2C, ...). Los datos que se generan por un servidor Web no son simplemente otra fuente de datos. Se puede decir que es un flujo de datos en continua evolución. En la actualidad existen más de doce formatos diferentes para capturar los datos de los servidores. Cada uno de estos formatos tienen componentes opcionales que si se usan podrían ayudar a identificar usuarios, sesiones y en ultima instancia comportamientos.

Sin embargo, las organizaciones necesitan construir la infraestructura para la revolución que ha supuesto la red Internet si quieren sobrevivir en el entorno competitivo en el que se manejan. El éxito de la mayoría de actividades que se van a desarrollar en los próximos años en la Web va a depender de cómo los servidores Web respondan a sus visitantes. Sólo entendiendo el flujo de datos generado por

los servidores Web se podrá dar un servicio personalizado a los usuarios. Entender este flujo de datos va a suponer construir la infraestructura de bases de datos para dar soporte a la toma de decisiones. Todo esto no se podrá conseguir sin una metodología que permita analizar mediante proyectos de *Data Mining* los grandes volúmenes de datos que se van a generar en los próximos años en los diferentes sitios web. Tal metodología tiene que contemplar no sólo los procesos de análisis en sí mediante las técnicas mas avanzadas de bases de datos, sino que tendrá que establecer los pasos a seguir para poder capturar la información adecuada que permita conocer quiénes son los visitantes y cuáles son sus pautas de comportamiento.

Con seguridad podemos afirmar que en los próximos años surgirán nuevas capacidades para almacenar y extraer información de los servidores web. El lenguaje XML y el lenguaje HTML dinámico han hecho que las páginas web sean más expresivas y personales, lo que afectará a los flujos de datos generados. Debido a la naturaleza distribuida de la web los datos se están recogiendo simultáneamente por servidores físicos diferentes. Incluso cuando los formatos de estos servidores son compatibles surge el problema de la sincronización de dichos ficheros. Por otra parte, se tienen datos generados por los proveedores de servicio de Internet (ISP), por los proxies, por los motores de búsqueda y se necesita que todos estos datos se analicen en el servidor que queremos estudiar. Otra de las grandes frustraciones con el flujo de datos generados es el anonimato de las sesiones. Esto es debido en parte a las características del protocolo `http` subyacente. Este problema se tiene que estudiar en profundidad para encontrar la mejor solución para poder identificar los usuarios de manera única. Para analizar los comportamientos no va a ser suficiente colocar cada suceso que ocurre en una página o cada gesto de un usuario en una base de datos. Esto tan solo nos proporcionaría una descripción inútil del comportamiento. Estos datos han de ser enriquecidos e interpretados para intentar encontrar perfiles de comportamiento, grupos de usuarios, valor potencial de los usuarios y todo de cara a poder proporcionar un servicio personalizado.

Por tanto, será necesario diseñar un entorno que permita recoger la información de forma correcta y que permita un posterior procesado de la misma de manera eficiente de tal forma que sea posible identificar correctamente a los usuarios, se puedan obtener las sesiones de manera automática y finalmente se pueda extraer conocimiento. No obstante todo ello necesita de una infraestructura en la que el almacenamiento y el posterior acceso a los datos se realice de manera eficiente.

Como consecuencia, para resolver el problema se plantea el diseño e implementación de un sistema que será capaz de:

- Recoger datos provenientes de la Web.
- Procesar los datos de cara a la obtención inteligente de sesiones e identificación de usuarios.
- Almacenar los datos en un soporte especialmente diseñado para este tipo de datos.
- Acceso eficiente a los datos almacenados.
- Análisis de los datos recogidos en tiempo real.

Los mayores problemas en el análisis de datos se deben principalmente a que no existe una arquitectura ni un soporte de datos adecuado. En primer lugar se trata de resolver el problema de la recogida de los datos y la identificación de sesiones y usuarios donde nos encontramos con el problema de la no persistencia del protocolo `http`. Tal problema se podría solucionar mediante la modificación de la recogida de los *logs* de los servidores. Una vez subsanado este problema se ha de disponer de un soporte de almacenamiento eficiente para poder acceder a grandes volúmenes de datos.

Se trata de que todo el entorno sea eficiente para lo que se plantea el diseño e implementación de una arquitectura de soporte a todas las operaciones derivadas del cálculo de perfiles web en tiempo

real. Las necesidades de cómputo y las prestaciones a nivel de entrada/salida requeridas por estas operaciones hacen necesaria la utilización de instalaciones hardware de altas prestaciones. El desarrollo de cluster basados en tecnología PC proporciona un rendimiento elevado a un coste muy inferior al de instalaciones basadas en grandes servidores centrales.

Además se hace necesario el estudio de los algoritmos de *Data Mining*. En la actualidad, los algoritmos que se están utilizando para el descubrimiento de información son los algoritmos de *Data Mining* que mejores resultados han demostrado en los últimos años. Sin embargo y debido al tipo y naturaleza de los datos con los que se está tratando se hace necesario el diseño e implementación de nuevos algoritmos. Como entorno de soporte al desarrollo de todas las operaciones derivadas de los nuevos algoritmos a tratar proponemos utilizar una arquitectura de agentes.

10.3. Bioinformática

Desde que en Febrero de 2001 *Celera Genomics* presentó en la revista *Science* el mapa del genoma humano, numerosos grupos de investigación en biología y genética han abordado ambiciosos proyectos dentro de esta área. Las oportunidades que brinda la biogenética en la detección precoz y tratamiento de gran número de enfermedades ha hecho de esta línea de investigación el campo más fructífero de la medicina experimental.

Las necesidades de cómputo de estos experimentos son muy elevadas, principalmente debido a dos factores; en primer lugar el tamaño de los datos que tratan, por ejemplo, el genoma humano consta de tres mil millones de bases, y en segundo lugar porque los algoritmos utilizados son cada vez más complejos y sofisticados lo cual implica la necesidad de recorrer los datos a analizar en varias ocasiones. Dentro de estos algoritmos se encuentran, por ejemplo, las técnicas de *Data Mining*, cuya aplicación a este campo se denomina *Bio Mining*. El soporte de computación requerido por estos experimentos implica un desembolso económico muy cuantioso en un computador con la potencia necesaria.

Las necesidades de los investigadores españoles que trabajan en genética o biología molecular a nivel de potencia de cálculo se han incrementado al mismo ritmo que han ido creciendo los trabajos realizados sobre dichos campos. Tal y como se recalcó en las II Jornadas Nacionales de Bioinformática, celebradas en Málaga a lo largo del mes de Junio de 2001, los experimentos de análisis de proteínas o secuencias de bases que se están realizando en centros como el CNB (Centro Nacional de Biotecnología) requieren el uso de grandes entornos de computación. El coste de una instalación de este estilo por medio de grandes computadores es excesivamente elevado y además sufren de unas severas limitaciones a la hora de su actualización o renovación.

Dentro de las áreas de interés de los investigadores en el campo de la bioinformática, se puede poner como ejemplo el agrupamiento de patrones de expresión génica. Para el primer caso se han utilizado algoritmos derivados de la inteligencia artificial, tales como las redes neuronales auto-organizativas [PPT+01], o estación de reglas de asociación (algoritmo Apriori) [RAPL+00]. Los algoritmos de este estilo realizan numerosas iteraciones sobre los datos a analizar, hasta lograr agrupar los elementos en diferentes clases de equivalencia. Si el conjunto de datos a analizar incluye miles de genes, y considerando que el tamaño medio de un gen es de 30000 bases, el tiempo de ejecución de los algoritmos y los recursos computaciones utilizados son muy elevados, por lo que la plataforma HPDA se puede adaptar perfectamente a este tipo de análisis, como ya veremos.

El creciente interés que en la actualidad se ha producido por el estudio del genoma, así como por toda la investigación relacionada con el análisis de datos genéticos en la previsión y detección de diversos tipos de enfermedades (hereditarias o degenerativas), impone la necesidad de combinar dos factores. Un elemento fundamental en este tipo de problemas, ya mencionado, es la necesidad de cómputo y de recursos informáticos. En último lugar y no por ello menos importante, los algoritmos

y técnicas de análisis de la información del entorno bioinformático juegan un papel determinante en el desarrollo y los objetivos del sistema integral.

10.3.1. Aplicación de algoritmos de *Data Mining* a datos biogenéticos

La explotación práctica de un sistema con estas características se basa en su adecuación a las operaciones y datos que van a tratar. Las características de los datos que afectan directamente a su procesamiento por el sistema se basan en dos factores; en primer lugar, en las técnicas aplicadas a los mismos y en segundo lugar, en el volumen y taxonomía de los propios datos.

1. Las técnicas habitualmente usadas son técnicas de estudio estadístico. Estas técnicas no consiguen nada más que una información global de parámetros estadísticos, tales como distribuciones o correlaciones entre valores. El estado del arte en técnicas de análisis de este tipo de datos apunta hacia el uso de algoritmos de *Machine Learning* o de *Data Mining*.

2. Las bases de datos de genómica o de síntesis de proteínas incluyen centenares de miles o incluso millones de registros, cada uno de los cuales dispone, a su vez de varios miles de atributos. Esto hace que su gestión en memoria principal de un computador sea en muchos casos imposibles.

3. La combinación del volumen de datos con los algoritmos tradicionales abre la puerta a multitud de problemas que no se dan en entornos con diferentes características. Algoritmos tales como la extracción de reglas de asociación mediante variantes del algoritmo Apriori de Agrawal, a la hora de trabajar con problemas de gran tamaño, presentan nuevas problemáticas. En este caso, por ejemplo, las propias estructuras intermedias de almacenamiento del algoritmo (los conjuntos de atributos candidatos en cada iteración en Apriori) exceden, incluso en varios órdenes de magnitud, el volumen de los datos a tratar.

10.4. HPDA

En la actualidad el incremento de potencia asociado a su reducido coste ha hecho de la tecnología PC una herramienta útil más allá de su aplicación inicial como terminal de trabajo. Sin embargo las prestaciones de un PC, incluso de última generación, distan mucho de las necesidades de cálculo y almacenamiento masivo de entornos como los de *Web Mining* o bioinformática. Como alternativa está siendo utilizado en numerosos problemas de super-computación la agrupación de computadores de tipo PC en redes o clusters. Estas redes de computadores tienen dos grandes ventajas; por un lado, la potencia de cálculo y almacenamiento combinada de todos sus elementos es equivalente a grandes instalaciones de un solo computador, teniendo además un coste apreciablemente menor; por otro lado, el diseño de un cluster de procesadores permite la escalabilidad y la actualización del entorno agregando o reemplazando sus componentes según lo exijan las necesidades del problema.

No obstante, los entornos de computación distribuida, como estos clusters de PC, son mucho más difíciles de explotar y utilizar que los entornos centralizados.

Entre las cuestiones más relevantes en el aprovechamiento de los entornos de procesamiento masivo se encuentran los problemas de almacenamiento o de entrada/salida distribuida. La distribución de los datos en los nodos de un cluster hace de su acceso una operación no transparente; sin embargo, proporciona un grado de escalabilidad y una oportunidad de paralelizar las operaciones de lectura/escritura sobre los elementos del cluster. Otro factor importante en la utilización de un cluster de computadores son los mecanismos para repartir y equilibrar la carga. Estos mecanismos deben permitir utilizar el número de nodos necesarios para minimizar el tiempo de ejecución de un conjunto de tareas. Las técnicas de equilibrado de carga deben evitar que se encuentren nodos ociosos mientras que otros

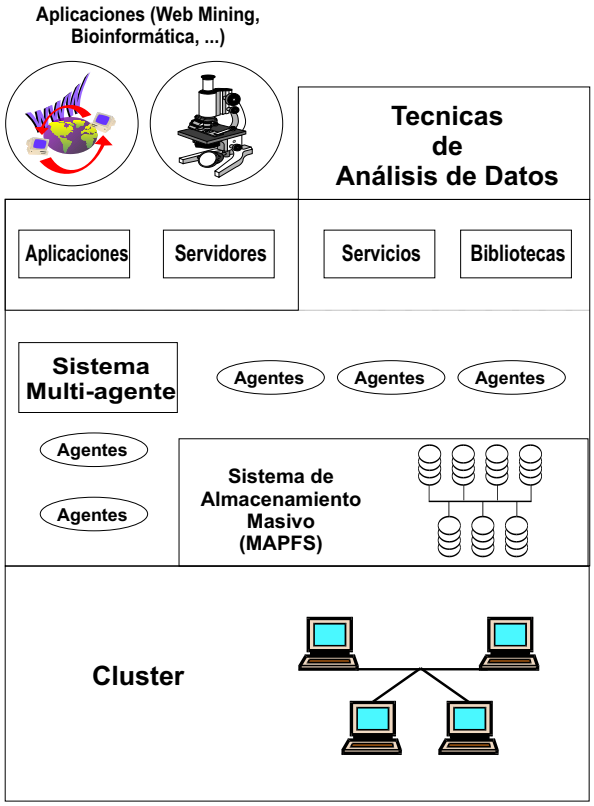


Figura 10.1: Arquitectura de HPDA

nodos de proceso se encuentran saturados. La resolución ideal a este problema debe permitir que los factores tanto de los procesos de entrada/salida como de la gestión de las tareas repartidas entre los nodos sean lo más transparente posible al investigador y sus algoritmos y técnicas. Esta idea debe mantener un equilibrio con el grado de aprovechamiento de los recursos del cluster. De poco sirve una capa que oculte completamente los problemas antes citados, pero que no utilice más que una pequeña fracción de las capacidades del cluster que recubre.

Como hemos mencionado anteriormente, HPDA da respuesta a estas necesidades y consiste en un sistema que integra varios niveles o capas, a fin de realizar un análisis de datos de una forma eficiente. El sistema es independiente del dominio de las aplicaciones, aunque en el desarrollo del prototipo se han diseñado dos capas que pueden utilizar este sistema y que permiten sacar partido de la infraestructura subyacente: una capa de aplicación para algoritmos de *Web Mining* y otra capa de aplicación para algoritmos de bioinformática.

Como consecuencia, HPDA consiste en una arquitectura que proporciona un soporte eficiente de la recogida, almacenamiento y el posterior análisis de los datos pertinentes. Para el acceso eficiente a los datos se hace necesario implementar un sistema de almacenamiento masivo al que se pueda acceder en paralelo. En este punto entra en juego el sistema de ficheros paralelo MAPFS. El entorno resultante pretende así eliminar los problemas de los actuales sistemas de recogida y análisis de datos. Además, el sistema permitirá el análisis en tiempo real de los datos recogidos, cuestión que tampoco es posible a menos que se disponga de un soporte adecuado.

La figura 10.1 muestra la arquitectura genérica del sistema en cuestión.

Debido a las características del sistema de ficheros MAPFS, el sistema multi-agente es compartido

por MAPFS y por las capas superiores que aprovechan los servicios ofrecidos por los agentes que lo forman.

La arquitectura propuesta para el desarrollo del sistema planteado se basa en un cluster de estaciones de trabajo de tipo PC de gama media con gran capacidad de almacenamiento. Por encima de este hardware se utiliza un sistema de almacenamiento paralelo (MAPFS) que permite la gestión y recuperación de la información con un alto grado de rendimiento. La configuración del entorno y la parametrización del sistema de ficheros paralelo se controla por medio de un sistema de agentes cooperantes. Sobre la base de los servicios proporcionados por estos elementos se apoyan los servidores de información convenientemente modificados que se utilicen para proporcionar el servicio y un conjunto de librerías y herramientas relativas al dominio de las aplicaciones. La información analizada por este entorno es proporcionada por los servidores anteriormente citados, analizada por medio de los algoritmos de análisis de datos (principalmente, *Data Mining*) y sus resultados y perfiles serán realimentados a los servidores de información para futuras utilizaciones.

10.5. Algoritmo Apriori

En el campo de *Data Mining* toman un papel destacado las denominadas **reglas de asociación**. Las reglas de asociación permiten modelizar patrones útiles e interesantes en muchos escenarios de *Data Mining*. Estos patrones representan características que "suceden juntas" y que son deducidas a partir de los datos. La aplicación más conocida de las reglas de asociación se conoce como "problema de la cesta de la compra", un problema genérico que trata con la identificación de los productos que realiza un cliente en la misma compra. El problema de la cesta de la compra define una clara tarea de reconocimiento de patrones, que es aplicable a muchos otros problemas, tales como la predicción de fallos en redes, análisis del genoma o tratamientos médicos, entre otros. Las reglas de asociación se utilizan en escenarios donde hay grandes bases de datos, los cuáles contienen millones de instancias.

Una regla de asociación es una implicación de la forma $A \rightarrow B$, donde A y B son conjuntos de datos llamados "elementos". Estos conjuntos no tienen elementos comunes ($A \bigcap B = \emptyset$) y no son vacíos ($A \neq \emptyset \wedge B \neq \emptyset$). La regla $A \rightarrow B$ debe leerse como "si una instancia cumple la característica A, entonces cumple también la característica B".

Una regla de asociación tiene dos parámetros numéricos que definen la calidad de la misma:

- *soporte de una regla de asociación:* El soporte de una regla R = $A \rightarrow B$ representa cuantas instancias son descritas por la regla, es decir, $soporte(R) = |A \bigcup B|$, donde $|X|$ es el número de instancias con la característica X.
- *confianza de una regla de asociación:* La confianza de una regla $R = A \rightarrow B$ es el soporte del antecedente de la regla dividido por el soporte de todos los elementos que aparecen en la regla, es decir, $confianza(R) = |B|/|A \bigcup B|$.

El algoritmo Apriori propuesto por Agrawal [AIS93] es el algoritmo más conocido de reglas de asociación. Este algoritmo tiene dos fases diferentes: (i) cálculo de conjuntos frecuentes y (ii) extracción de reglas utilizando estos conjuntos frecuentes. La primera de estas dos fases se convierte en un "cuello de botella" en el algoritmo Apriori. En esta primera fase es necesario acceder al sistema de almacenamiento para obtener una gran cantidad de datos que deben procesarse. El acceso al sistema de E/S es lento y los sistemas tradicionales no incrementan el rendimiento de las operaciones de E/S. El algoritmo Apriori calcula los conjuntos frecuentes en varias fases iterativas en cada una de las cuáles el algoritmo recorre la base de datos entera una vez. Cada iteración comienza con cada grupo de conjuntos de datos candidatos que son validados después de que la base de datos entera es leída y los soportes de los conjuntos calculados.

Una optimización posible de este algoritmo es el cálculo paralelo de estos conjuntos de datos frecuentes, distribuyendo los conjuntos de datos candidatos entre diferentes nodos. Cada uno de los nodos recorre la base de datos entera para calcular el soporte de sus propios candidatos. Los algoritmos *Data Distribution* [AS96] e *Intelligent Data Distribution* [HKK97] son ejemplos de este tipo de algoritmos.

Por otro lado, se puede también llevar a cabo lo que se denomina paralelismo de datos, particionando la base de datos entre los diferentes nodos. Con esta técnica, cada uno de los nodos sólo recorre sus datos locales, calculando el soporte de cada candidato y posteriormente combinando sus resultados con los del resto de los nodos. PEAR [Mue95], FDM [CHN+96] y NPA [SK96] son algoritmos que permiten hacer el cálculo del soporte de forma paralela.

También existen algoritmos híbridos que combinan paralelismo de cálculo y de datos. Shintani y Kitsuregawa [SK99] presentan varios algoritmos, que distribuyen tanto los datos como los conjuntos de candidatos para mejorar el cálculo de los soportes.

Otra alternativa es incrementar el rendimiento reduciendo el tamaño de la base de datos después de cada recorrido de la misma. DHP [PCY95] elimina registros de la base de datos cuando el número de candidatos es menor que el tamaño del próximo conjunto de datos. PDM [CHN+96] es una implementación paralela de este algoritmo.

Otros algoritmos paralelos han sido propuestos teniendo en cuenta arquitecturas hardware o características especiales. Zaki et al. diseñaron algoritmos eficientes para arquitecturas con memoria compartida [ZOPL96] o clusters de SMP [ZPOL97].

Como aplicación de MAPFS se presenta una optimizacion del algoritmo Apriori, haciendo uso de dicho sistema de ficheros y distribuyendo los datos a través de un cluster de nodos [PPG+02a]. MAPFS proporciona acceso paralelo a los datos, reduciendo el cuello de botella que constituye el acceso a servidores convencionales. Para ello, se ha modificado ligeramente el algoritmo Apriori de forma que utilice la interfaz de MAPFS.

Como ya hemos mencionado, algunos algoritmos eliminan registros en etapas iniciales, de forma que reducen el tamaño del problema. Otros algoritmos comienzan con el conjunto completo de atributos y durante las múltiples iteraciones el conjunto se va reduciendo. Si determinados atributos se marcan como eliminados, una estrategia interesante sería saltar su posición en futuras lecturas. Esta es la técnica utilizada en esta optimización. El sistema de almacenamiento es el responsable de descartar bloques de datos cuyos elementos no son candidatos para las siguientes etapas del algoritmo. La idea principal es utilizar la información generada en una iteración para reducir la entrada de datos de la próxima iteración.

Véamos como se lleva a cabo la optimización. Sea I el conjunto de todos los conjuntos de elementos. El algoritmo Apriori se puede definir del siguiente modo:

1. Se calculan las ocurrencias de elementos que determinan los mayores conjuntos de datos de 1 elemento, que denominaremos L_1.

2. Se lleva a cabo el siguiente bucle:

for (k=2; $L_{k-1} \neq \emptyset$; k++) {
/* Generación de nuevos candidatos */
C_k= *candidates_gen*(L_{k-1});
forall itemsets i in I do
/* Obtener candidatos de i */
$C_i = subset(C_k, i)$
forall candidates c in C_i do
c.count++;
$L_k = \{c \in C_k / c.count >= min\_support\}$
}

3. La salida es: $\bigcup_k (L_k)$.

donde *candidates_gen()* es una función que acepta el conjunto de todos los mayores conjuntos de datos de $k-1$ elementos L_{k-1}, y devuelve un superconjunto de todos los mayores conjuntos de datos de k elementos. Estos conjuntos de datos permiten encontrar reglas de asociación entre diferentes elementos.

Por otro lado, el parámetro *min_support* se define para garantizar que las reglas son significativas.

Para optimizar el algoritmo Apriori, se ha utilizado la implementación de Christian Borgelt, que utiliza un árbol de prefijos para organizar los contadores de los conjuntos de elementos [CHR]. Esta implementación utiliza ficheros planos como datos de entrada y ficheros de salida que contienen las reglas de asociación junto a su confianza y soporte. La optimización se realiza a nivel del sistema de ficheros. El sistema de almacenamientos se encarga de descartar en la iteración i aquellos bloques de datos que no sean miembros de los candidatos en el nivel $i-1$.

Esta propuesta utiliza *hints*, construidos mediante un subsistema multiagente que permite incrementar el rendimiento de la lectura de la información. La implementación de Christian Borgelt lee todas las líneas de fichero de entrada en cada iteración. De cara a incrementar el rendimiento, el bloque escogido es una línea del fichero de entrada. El sistema de almacenamiento utiliza *hints* que definen si un bloque (una línea) debe evaluarse en la próxima iteración o no. Los bloques se marcan como eliminados si no han sido seleccionados en la iteración anterior. Si un bloque está marcado como eliminado, el sistema de ficheros salta este bloque en la lectura y de este modo, se reducen los datos de entrada.

La implementación de Christian Borgelt se ha modificado básicamente en dos aspectos:

1. Utiliza las operaciones de E/S de MAPFS.
2. Interacciona con MAPFS para comunicar al sistema de almacenamiento que un conjunto de elementos pertenece a los candidatos. Si para algún conjunto de elementos, no se "avisa" al sistema de almacenamiento de que dicho conjunnto pertenece a los candidatos, este bloque se marca como eliminado.

Por tanto, las mejoras llevadas a cabo sobre la implementación son básicamente dos:

1. acceso paralelo a los datos;
2. uso de *hints* para descartar bloques de datos.

La sección 11.6 recoge la evaluación de esta modificación, que permite aumentar la eficiencia de una aplicación ya bastante mejorada.

10.6. Resumen

Este capítulo describe la interacción entre diferentes aplicaciones y el sistema de ficheros MAPFS, como plataforma subyacente para el acceso a los datos. Como ejemplo de interacción se describe el sistema HPDA (*High Performance Data Analysis*), sistema que permite llevar a cabo un análisis de grandes cantidades de datos, logrando conseguir un alto rendimiento. HPDA utiliza MAPFS como sistema de ficheros. A pesar de que HPDA se ha concebido como un entorno independiente de aplicación, dicho sistema va a ser utilizado básicamente por dos tipos de aplicaciones: aplicaciones de *Web Mining* y aplicaciones de bioinformática, que necesitan acceder a una gran cantidad de información.

Además del análisis del sistema HPDA, en este capítulo se ha procedido a describir la mejora realizada sobre el algoritmo Apriori. Esta implementación permite que el sistema de almacenamiento tome decisiones sobre los datos de entrada, mejorando la eficiencia del algoritmo a través del acceso paralelo a los datos y el uso de *hints*.

Capítulo 11

Evaluación de MAPFS

11.1. Introducción

Los sistemas deben ser medidos o evaluados para conocer las deficiencias y ventajas que presentan. Aunque una evaluación cualitativa puede resultar útil para validar un sistema, es necesario cuantificar los resultados a fin de comparar el rendimiento de la solución. Un sistema validado cualitativamente puede no cumplir requisitos cuantitativos que le faculten para su funcionamiento real.

Por otro lado, hay una tendencia a evaluar los sistemas en condiciones idóneas y por tanto, en situaciones no reales, bien sea porque las medidas se realizan en períodos de tiempo donde no hay carga de trabajo, bien sea porque el entorno de pruebas no se corresponde con el entorno en el cuál el sistema va a ser implantado o bien porque se utilice una herramienta de simulación. Pero estas medidas no se adaptan al funcionamiento real.

Por este motivo, en este capítulo se recogen las evaluaciones realizadas sobre el sistema de ficheros MAPFS sobre un entorno real de trabajo y con aplicaciones o herramientas reales, a fin de justificar la utilidad de dicho sistema.

11.2. Evaluación de un sistema

Para evaluar un sistema, debemos tener en cuenta que la eficiencia está limitada por la *Ley de Amdahl* [Amd67], [SBN82]. Esta ley afirma que *"la mejora obtenida en el rendimiento al utilizar algún modo de ejecución o componente más rápido está limitada por la fracción de tiempo que se pueda utilizar ese modo o componente más rápido"*. Es decir,

$$S_{eff} = \frac{S_f}{S_f(1-f)+f}$$

donde:

- *f:* es la fracción del sistema que puede ser mejorada;

- S_f*:* el factor de mejora y
- S_{eff}*:* es la mejora global lograda o *speedup* del sistema.

Obviamente, para un valor pequeño de *f*, $S_{eff} \approx 1$, cualquiera que sea el valor de S_f. Es decir, la mejora global lograda es insignificante.

Un aspecto importante que se deriva de la ley de Amdahl es el hecho de que *"el speedup obtenido por un sistema está limitado por el componente más lento del sistema"*.

La ley de Amdahl aplicada al campo del paralelismo se puede expresar de la siguiente forma: *dada una aplicación paralela que puede ser ejecutada en n procesadores interconectados, el speedup logrado por la misma viene dado por la siguiente expresión:*

$$S_{eff} = \frac{n}{n * (1 - f) + f} = \frac{1}{(1 - f) + f/n} \tag{11.1}$$

Para este cálculo, se han tenido en cuenta dos simplificaciones:

1. Se asume el caso ideal de que N trabajadores pueden hacer el trabajo en una fracción $1/N$ del tiempo que necesita un trabajador.
2. Se ignora el coste de comunicación.

Según la expresión 11.1, el valor máximo del *speedup* está limitado por el número de procesadores. Esta es la cota máxima, salvo en determinadas situaciones en las que se produce un fenómeno denominado *speedup superlineal*, que consiste en que el aumento en el número de procesadores también implica una mejora de otros factores, como por ejemplo un incremento de la cache global.

Otro índice relacionado con la métrica del rendimiento en un sistema paralelo es la *eficiencia*.

La eficiencia es una medida de la fracción de tiempo que un procesador está siendo utilizado. Se define como la razón entre el *speedup* y el número de procesadores:

$$E = \frac{S_{eff}}{n}$$

Este índice oscila entre 0 y 1 (salvo en casos de *speedup* superlineal).

11.3. Evaluación de MAPFS. Pruebas de casos

Para evaluar el sistema de ficheros MAPFS, se han utilizado tres tipos de aplicaciones:

1. Operaciones de E/S: Se van a denominar operaciones de E/S en este apartado a aquéllas que se utilizan para la transformación de ficheros convencionales en ficheros MAPFS y viceversa, así como operaciones comunes entre ficheros.
2. Operaciones que explotan el paralelismo en el servidor: Cuando se hace referencia a paralelismo en el servidor, nos referimos al paralelismo que proporciona la distribución de los datos en los servidores y el acceso en paralelo a cada uno de ellos. Cualquier operación realizada en el sistema de ficheros MAPFS utiliza este tipo de paralelismo. Este tipo de aplicaciones permite que un único cliente pueda acceder a información distribuida en un conjunto de servidores de forma más eficiente que a dicha información situada en local (en un único nodo y accediendo mediante los servicios UNIX), si se utilizan determinados tamaños de acceso.

3. Operaciones que explotan el paralelismo en el cliente: Si se paraleliza un determinado algoritmo, se puede llevar a cabo la ejecución de `n` copias del algoritmo en la parte cliente. De este modo, la aplicación se beneficia del paralelismo en el servidor, inherente en el sistema de ficheros de MAPFS, y del paralelismo específico de la aplicación cliente.

Para las evaluaciones, se ha utilizado principalmente un cluster de 4 procesadores AMD Athlon a 650 MHz, discos SCSI, memoria principal de 256 MB y memoria cache de 512 KB unidos por una red Gigabit. El sistema operativo instalado en los PC es Linux v. 2.4.7-10. Como entorno principal, se ha utilizado un único grupo de almacenamiento, construido sobre cuatro servidores de NFS. Las pruebas han consistido en la evaluación del rendimiento de los tres tipos de aplicaciones anteriores sobre ficheros de distintos tamaños, utilizando también diferentes tamaños de acceso (1KB, hasta 512 KB). Las pruebas se han realizado en un entorno no dedicado y en horas de trabajo. Las medidas se han calculado como el valor media de 10 ejecuciones.

Respecto a la evaluación de los grupos de almacenamiento, se ha analizado el funcionamiento del sistema utilizando dos grupos con un número de nodos y características físicas diferentes, a partir de un escenario en el que se añade un nodo a un determinado grupo de almacenamiento. Asimismo, se ha calculado el tiempo de reconstrucción de los datos en el caso de uso de una operación de defragmentación. Uno de los grupos de almacenamiento es el anteriormente citado y el otro está formado por un conjunto de 8 nodos biprocesadores Intel Xeon a 2.40GHz, discos SCSI, memoria principal de 900 MB y memoria cache de 512 KB unidos por una red Gigabit. El sistema operativo instalado en este último cluster es Linux 2.4.20.

Para estudiar cómo afecta el número de nodos de E/S al rendimiento de la aplicación, también se ha llevado a cabo una evaluación del sistema con un número de nodos que oscila entre 2 y 8 nodos, utilizando el cluster de 8 nodos.

Antes de llevar a cabo la evaluación, es interesante caracterizar el sistema empleado adecuadamente. Para ello, se han evaluado la red y los discos de cada nodo, que son los recursos que limitan y condicionan el rendimiento del sistema.

A continuación se pasa a describir la evaluación de estos recursos en los dos clusters utilizados en las pruebas. Posteriormente, se muestran cada una de las evaluaciones de las diferentes aplicaciones sobre el sistema de ficheros MAPFS.

11.4. Caracterización del entorno de pruebas

El sistema de ficheros MAPFS utiliza principalmente dos recursos del cluster de nodos, la red, debido a su característica distribuida y los discos, debido a que utiliza este tipo de dispositivos como sistema de almacenamiento de los ficheros.

Para evaluar la red, se ha utilizado la herramienta `NetPIPE` (*Network Protocol Independent Performance Evaluator*) [TC02], que encapsula las mejores características de las herramientas de evaluación `ttcp` y `netperf`, permitiendo representar el rendimiento de la red bajo una variedad de condiciones. Esta herramienta realiza pruebas tipo *ping-pong*, enviando mensajes de tamaño variable e incremental entre dos procesos a través de la red o bien dentro de una arquitectura de memoria compartida.

Los resultados obtenidos para el cluster de 4 nodos se representan en la figura 11.1. Como se puede observar, el ancho de banda máximo es inferior a $30MB/s$.

Por otro lado, la figura 11.2 muestra los resultados de evaluación de la red para el cluster de 8 nodos. Destaca el hecho de que el ancho de banda máximo es aproximadamente $112MB/s$, muy superior al valor del cluster de 4 nodos.

Respecto a la evaluación de los discos, se ha utilizado la herramienta `hdparm` para medir el ancho de banda de dichos dispositivos. Después de un conjunto de pruebas de evaluación, se ha concluido

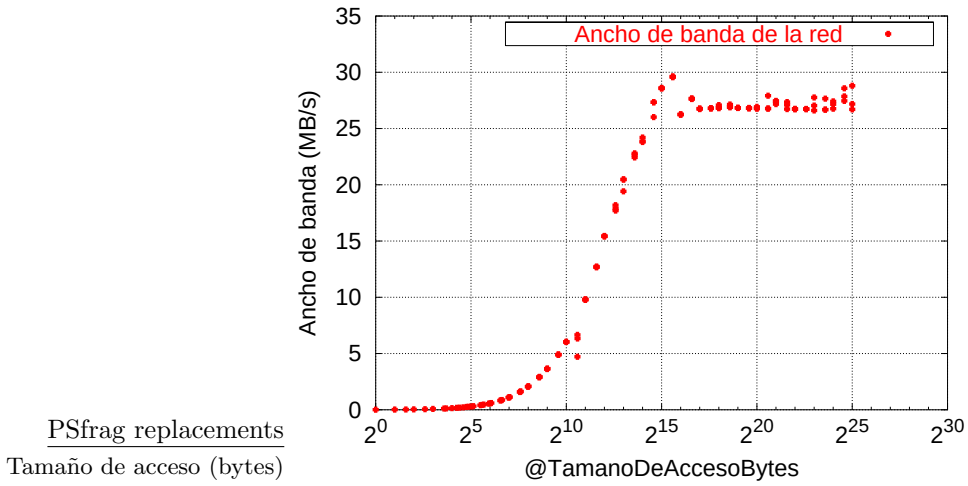


Figura 11.1: Evaluación del ancho de banda de la red del entorno de pruebas (Cluster de 4 nodos)

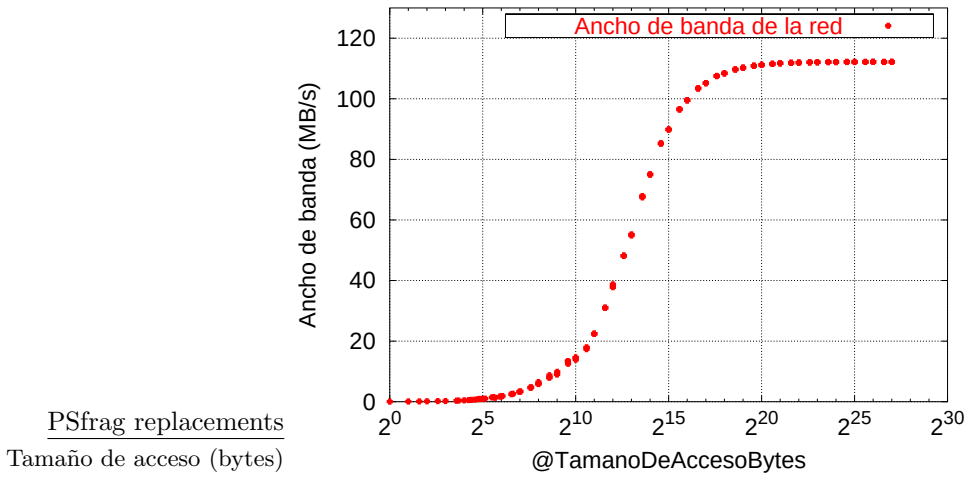


Figura 11.2: Evaluación del ancho de banda de la red del entorno de pruebas (Cluster de 8 nodos)

que el ancho de banda de los discos correspondientes a los 4 nodos del primer cluster tiene el valor medio de 25, 8045 MB/s, obteniendo una varianza de las medidas realizadas respecto a la media de 0, 00057475. Respecto al cluster de 8 nodos, la herramienta muestra que el valor medio del ancho de banda de los discos es de 46, 7150 MB/s, siendo el valor de la varianza respecto a la media de 0, 99980700. De nuevo, se puede observar un ancho de banda superior en este último caso.

11.5. Evaluación de las operaciones de E/S

Para el desarrollo de las evaluaciones, así como para lograr la integración entre un sistema de ficheros tradicional y el sistema de ficheros MAPFS, se ha implementado el siguiente conjunto de herramientas:

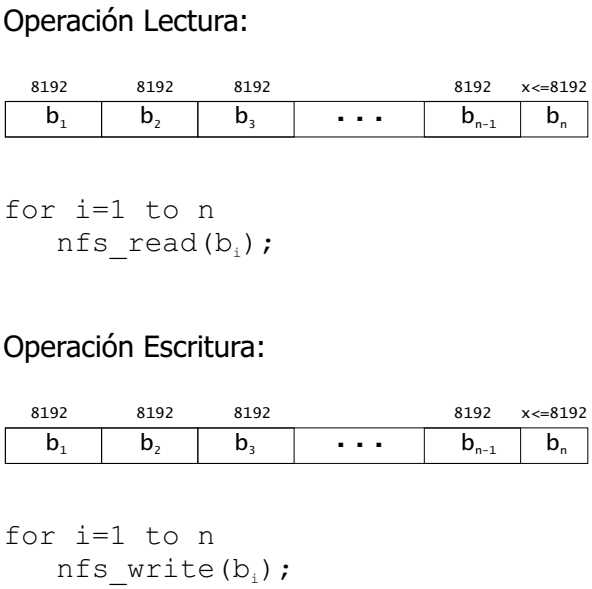


Figura 11.3: Agrupación de operaciones de lectura y escritura

- `pwrite`: Esta herramienta permite distribuir los datos de un fichero de un sistema de ficheros tradicional a través de un grupo de almacenamiento.

$$pwrite(f_t) = f_{MAPFS}$$

donde:

f_t : fichero tradicional

f_{MAPFS} : fichero MAPFS

- `pread`: Esta operación permite realizar la función inversa a `pwrite`, es decir, construir un fichero tradicional a partir de un fichero MAPFS.

$$pread(f_{MAPFS}) = f_t$$

- `pcopy`: Esta herramienta permite copiar un fichero MAPFS en otro fichero MAPFS. Se utiliza para evaluar el rendimiento de las operaciones de lectura y escritura de MAPFS.

$$pcopy(f_{MAPFS}) = f2_{MAPFS}$$

- `pcat`: Esta herramienta se ha utilizado exclusivamente para la evaluación de la operación de lectura MAPFS `mapReadContig()`. La operación consiste en la lectura de un fichero MAPFS, descartando su contenido.

- `pflush`: De forma análoga a la herramienta anterior, ésta se utiliza sólo para evaluar la operación de escritura MAPFS `mapWriteContig()`. La operación consiste en la escritura de un determinado *buffer* en un fichero MAPFS.

Tras las pruebas realizadas en una fase inicial, se dedujo que las operaciones de lectura y escritura MAPFS tienen mejor rendimiento en los siguientes casos:

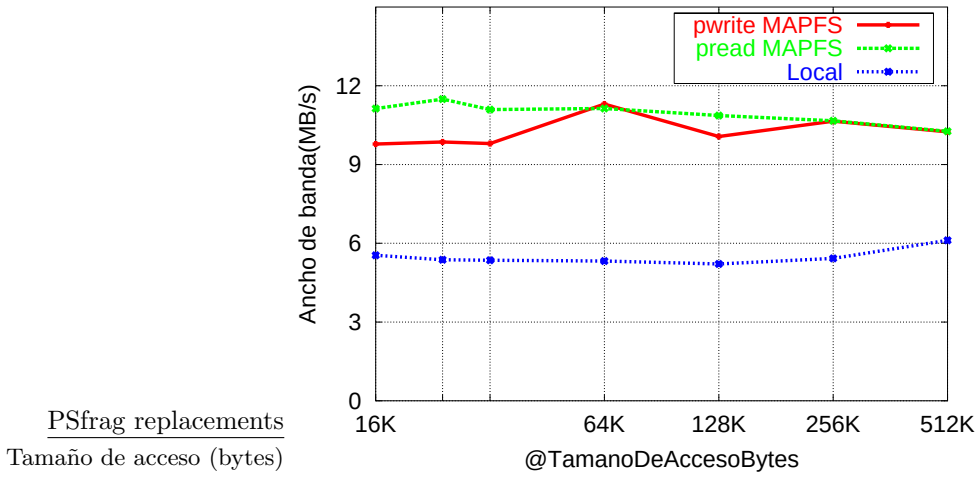


Figura 11.4: Evaluación de las operaciones de conversión entre ficheros convencionales y ficheros MAPFS (`pwrite` y `pread`) con ficheros de 150 MB

- A medida que aumenta el tamaño del fichero.
- A medida que se incrementa el tamaño de acceso. Esta tendencia disminuye a partir de un determinado tamaño, que depende de la operación realizada y del tamaño del fichero. No obstante, NFS tiene una limitación en las operaciones de escritura y lectura `nfsproc_read_2` y `nfsproc_write_2`[1], ya que sólo permite leer y escribir un máximo de 8192 bytes [NFS89]. Para eliminar esta limitación, las operaciones que se realicen con un tamaño de bloque superior, MAPFS las trata como una agrupación de operaciones de tamaño máximo. En la figura 11.3 se muestra la forma de abordar la agrupación de operaciones de lectura y escritura.
- En aplicaciones que tienen una significativa proporción de operaciones de E/S frente a la fase de procesamiento.
- Cuando el sistema se utiliza de un modo paralelo, es decir, no se mezclan operaciones de E/S paralelas y secuenciales y el procesamiento de la información también se lleva a cabo de una forma paralela. Cuando se secuencia alguna de ellas, puede ocurrir que la fase secuencial se convierta en "cuello de botella" de la fase paralela. Precisamente a esta característica es a la que se ha denominado *paralelismo en el cliente*. Éste es uno de los motivos por los que el sistema de ficheros MAPFS se ha portado a MPI, de forma que la fase de procesamiento se pueda paralelizar e integrar de una forma sencilla y estándar.

Debido a que las operaciones de E/S utilizadas para la evaluación sólo explotan el paralelismo del servidor, se observó que para tamaños de acceso pequeños (menores de 16 KB), los tiempos de dichas operaciones eran peores que las homólogas en local, debido principalmente al excesivo número de operaciones de E/S, que obligan a utilizar de forma intensiva la red. A partir de un tamaño de acceso de 16 KB, el rendimiento de las operaciones de E/S se incrementa frente al acceso local.

Las figuras 11.4, 11.5, 11.6 y 11.7 muestran los resultados de evaluación de las operaciones de E/S, a partir de este tamaño de acceso y para ficheros de 150 MB.

Para comprobar que esta tendencia se mantiene en el caso de ficheros con un tamaño mayor, se han realizado las mismas pruebas para ficheros de tamaño 300 MB y pruebas de lectura y escritura

[1] operaciones que utiliza NFS para llevar a cabo la lectura y escritura respectivamente

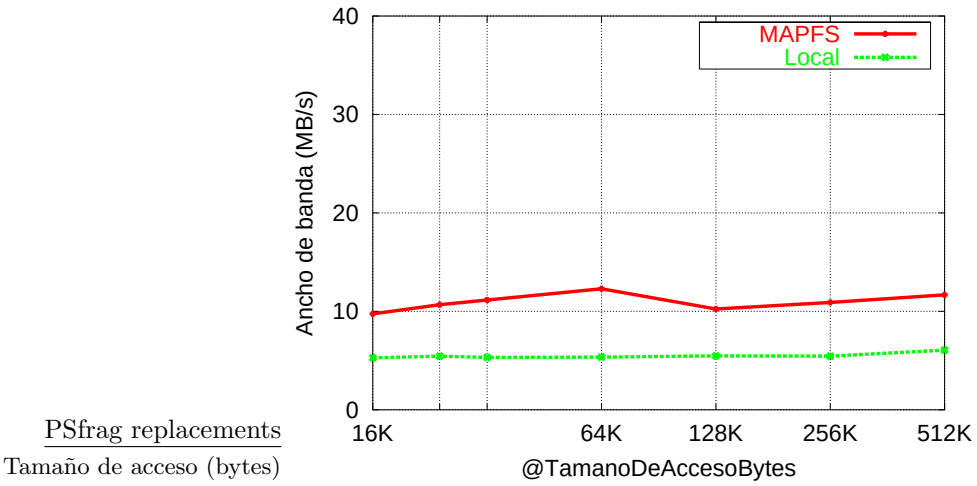


Figura 11.5: Evaluación de la operación de copia de ficheros (`pcopy`) con ficheros de 150 MB

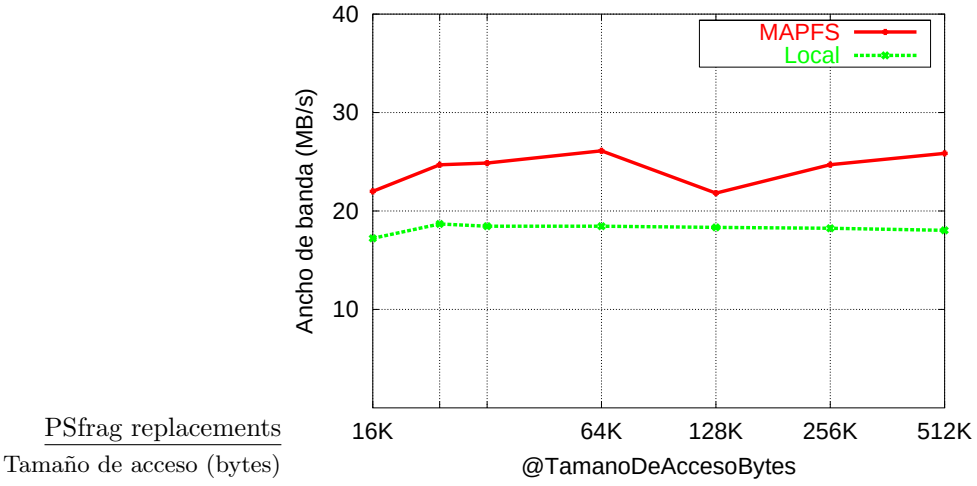


Figura 11.6: Evaluación de la operación de lectura (`pcat`) con ficheros de 150 MB

para ficheros de 1 GB. Los resultados se muestran en 11.8, 11.9, 11.10, 11.11 y 11.12. Estas últimas pruebas también se han realizado en el segundo cluster. La figura 11.13 muestra el resultado de esta evaluación.

Otro efecto que se ha analizado en este trabajo ha sido la influencia del número de nodos de E/S utilizados. Para ello, se ha utilizado el segundo cluster. La figura 11.14 muestra el tiempo de ejecución de la operación de escritura respecto al número de nodos de E/S y el tamaño de acceso. Como se muestra en dicha figura, el tiempo de ejecución decrece a medida que aumenta el número de nodos de E/S, debido al incremento del paralelismo de servidor.

Todas estas evaluaciones se han realizado sin hacer uso del subsistema multiagente, que permite optimizar las operaciones de lectura y escritura MAPFS mediante el uso de *prefetching* y *caching*. Para comprobar que el sistema multiagente permite mejorar estas operaciones, se ha llevado a cabo una evaluación de la operación de lectura, mediante uso de un sistema multiagente que proporciona la característica de *prefetching* secuencial con una ventana de 64 Kbytes y utilizando tamaños de bloque

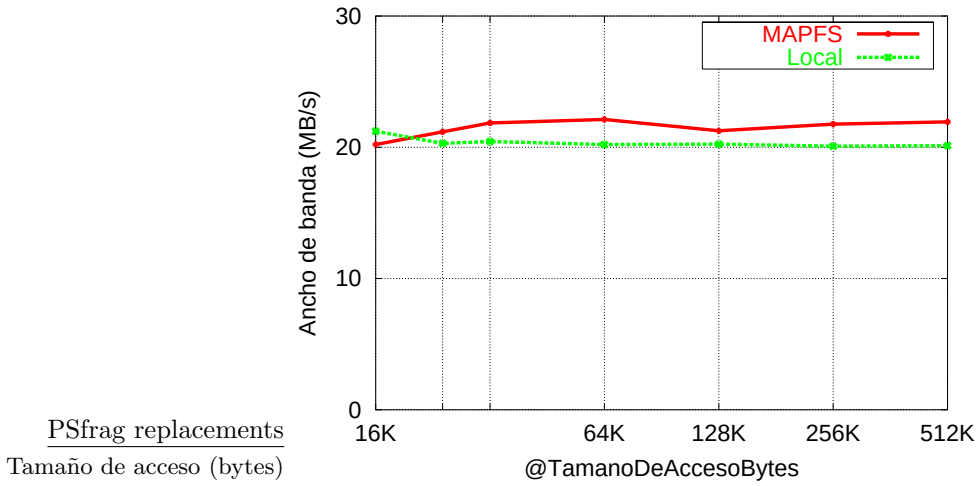


Figura 11.7: Evaluación de la operación de escritura (`pflush`) con ficheros de 150 MB

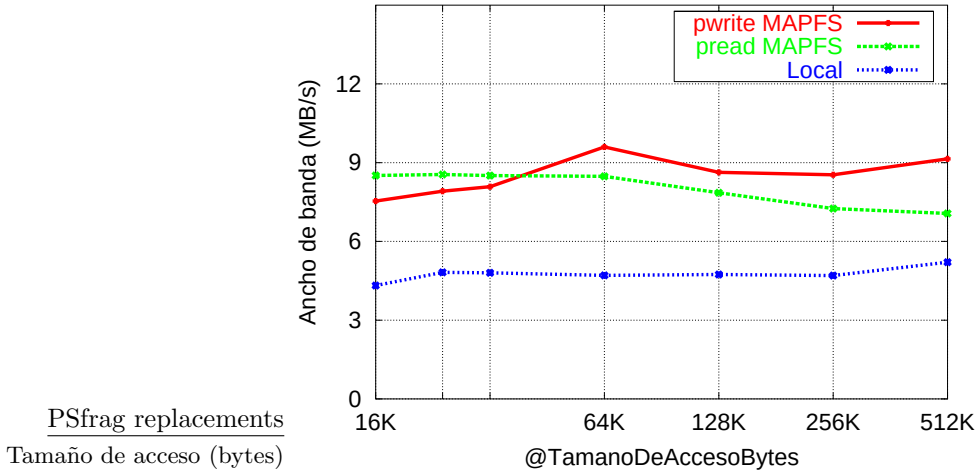


Figura 11.8: Evaluación de las operaciones de conversión entre ficheros convencionales y ficheros MAPFS (`pread` y `pwrite`) con ficheros de 300 MB

menores (entre 1 KB y 8 KB), que proporcionan rendimientos peores sin el uso de dicho subsistema. La figura 11.15 muestra dicha evaluación.

Por tanto, las conclusiones que se pueden deducir de las evaluaciones realizadas son:

1. El sistema MAPFS mejora el rendimiento de las operaciones de lectura y escritura locales, utilizando un único cliente. Para tamaños de acceso pequeños (entre 1 KB y 8 KB) es necesario utilizar el sistema multiagente *proxy* para lograr esta mejora.

2. Las operaciones que mezclan operaciones de lectura y escritura (`pread`, `pwrite` y `pcopy`) tienen un mejor *speedup* frente a las operaciones locales que las operaciones de un único tipo (`pcat` y `pflush`). Esto es debido a que como sólo se utiliza paralelismo en la parte del servidor, la única forma de permitir aumentar el paralelismo global es intercalando fases de E/S. La figura 11.16 compara el *speedup* de las operaciones de E/S.

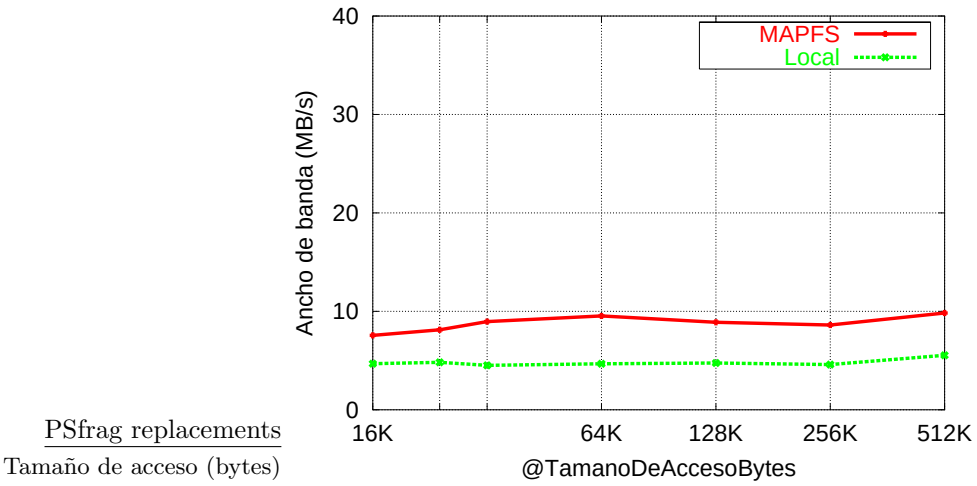


Figura 11.9: Evaluación de la operación de copia de ficheros (`pcopy`) con ficheros de 300 MB

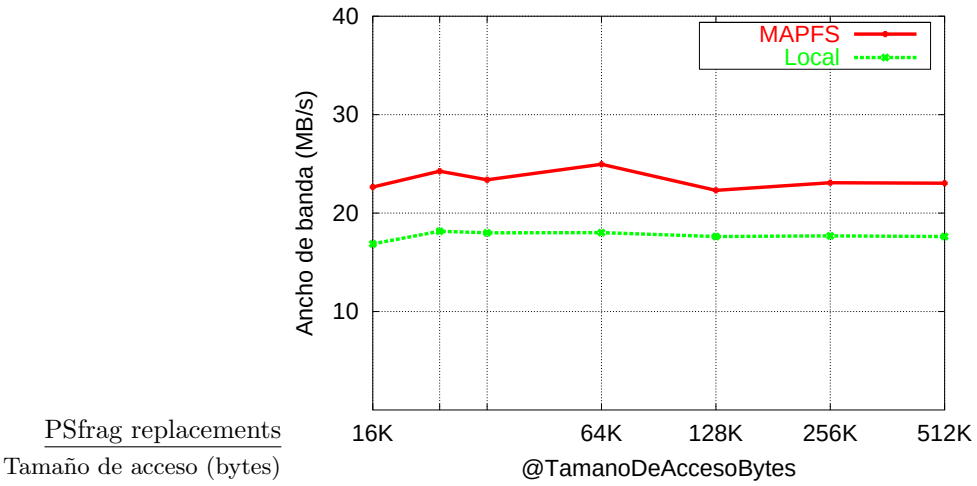


Figura 11.10: Evaluación de la operación de lectura (`pcat`) con ficheros de 300 MB

3. La relación existente entre el rendimiento de las operaciones de E/S y el tamaño de acceso utilizado se mantiene con diferentes tamaños de fichero de entrada.

11.6. Evaluación de operaciones que explotan el paralelismo en el servidor

De cara a utilizar una aplicación real que utilice el sistema de ficheros MAPFS y evaluar su rendimiento, se ha implementado una optimización del algoritmo Apriori (véase sección 10.5). Esta optimización permite beneficiarse de dos ventajas del sistema de ficheros MAPFS:

- Acceso paralelo a los datos.
- Uso de *hints*, que permiten descartar líneas en sucesivas iteraciones.

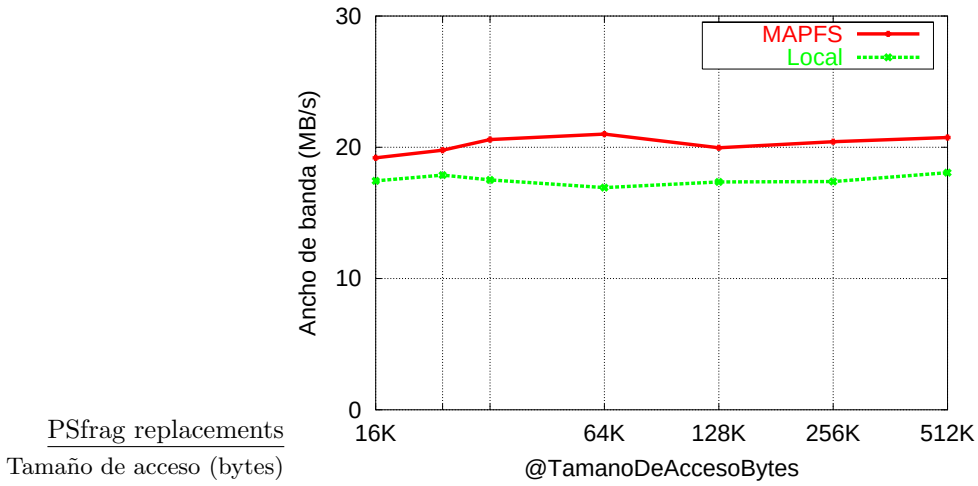


Figura 11.11: Evaluación de la operación de escritura (`pflush`) con ficheros de 300 MB

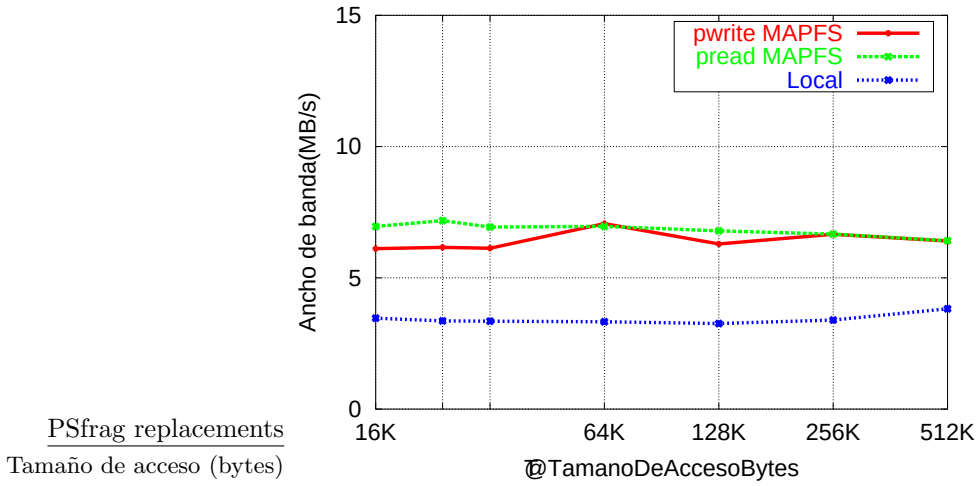


Figura 11.12: Evaluación de las operaciones de conversión entre ficheros convencionales y ficheros MAPFS (`pread` y `pwrite`) con ficheros de 1 GB

Para llevar a cabo la evaluación del sistema, hemos utilizado ficheros de entrada de 150 MB que contienen 131072 *items* por línea, generados sintéticamente a partir de un conjunto de un número de posibles *items* que oscila entre 35 y 50. Por su parte, dado un soporte de 1, el número de reglas generadas a partir de este tipo de ficheros oscila entre 1500 y 3000.

En la figura 11.17 se muestra la diferencia de tiempo de ejecución del algoritmo Apriori utilizando el sistema de ficheros MAPFS que accede a los datos de una forma distribuida y un sistema de ficheros tipo UNIX que accede a los datos en local.

Como hemos mencionado, una de las mejoras implementadas por el sistema de ficheros MAPFS es el uso de *hints* para optimizar el acceso a los datos. En el caso del algoritmo Apriori, los *hints* nos permiten descartar líneas del fichero de entrada en las sucesivas iteraciones de dicho algoritmo, realizando por tanto un acceso a los datos selectivo. En la figura 11.18 se muestra el porcentaje de líneas descartadas para diferentes valores de soporte.

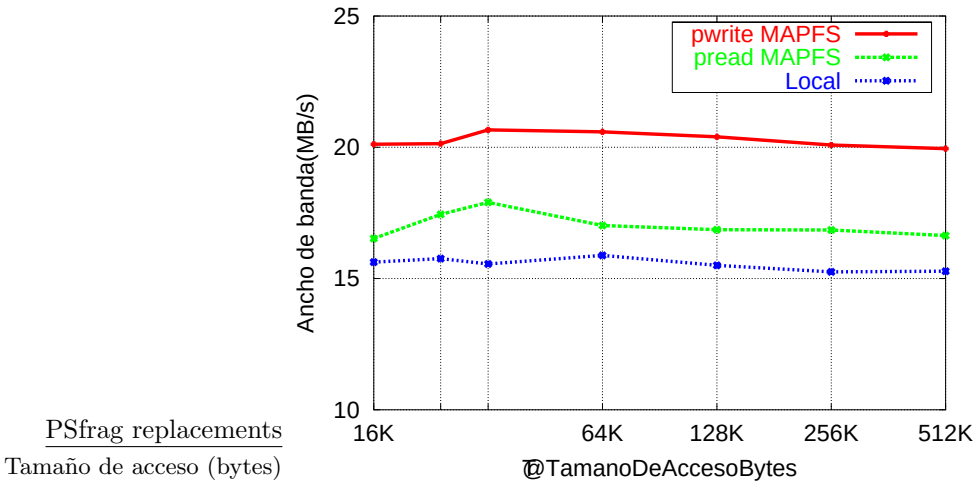


Figura 11.13: Evaluación de las operaciones de conversión entre ficheros convencionales y ficheros MAPFS (`pread` y `pwrite`) con ficheros de 1 GB en el cluster de 8 nodos

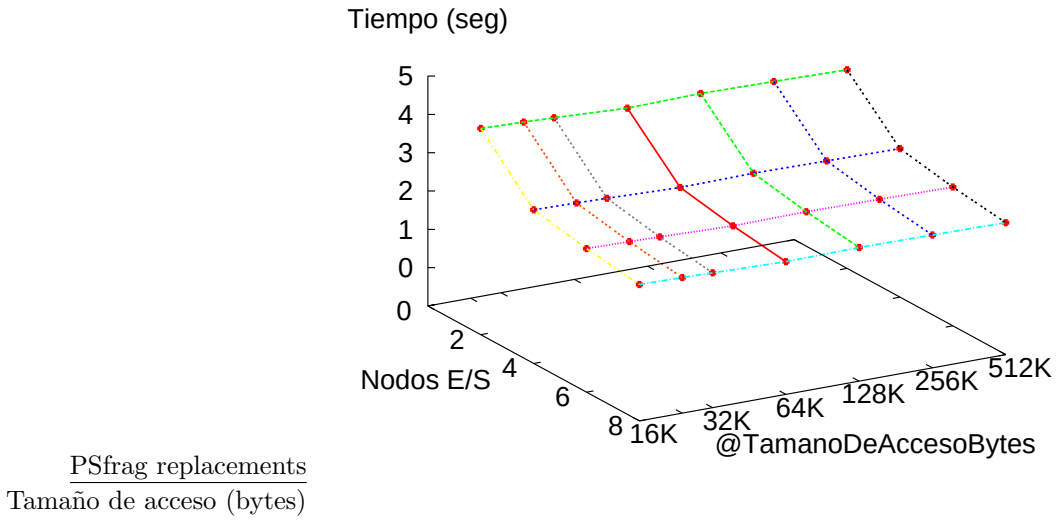


Figura 11.14: Influencia del número de nodos de E/S utilizados

Este proceso implica optimizar la ejecución del algoritmo. La figura 11.19 muestra los tiempos de ejecución del algoritmo con y sin *hints*. Aunque la figura no muestra unas diferencias muy notables, la optimización es considerable, por los siguientes motivos:

1. A medida que se incrementa el tamaño del fichero, la diferencia es mayor. En ficheros muy grandes, esta diferencia en tiempo de ejecución es muy apreciable.

2. El algoritmo Apriori utilizado como base [CHR] ya está muy optimizado, por lo que la mejora lograda es muy significativa.

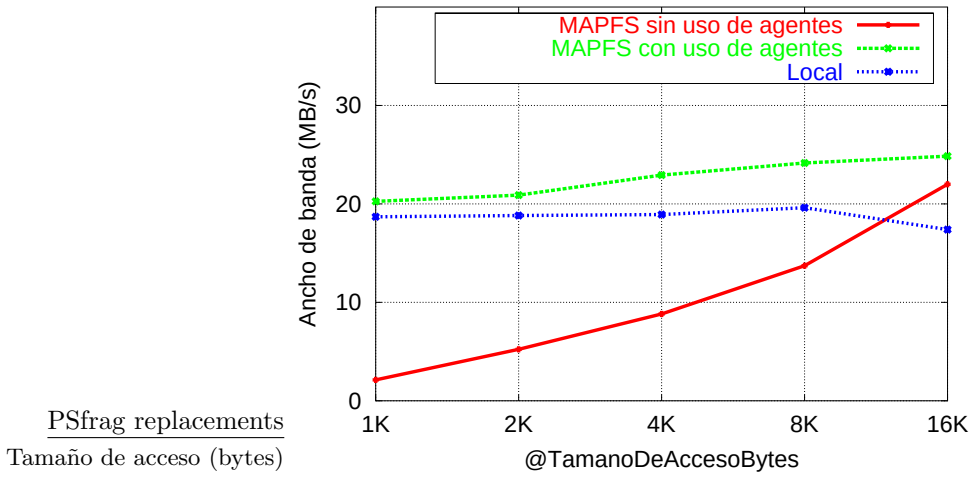


Figura 11.15: Evaluación de la operación de lectura (`pcat`) con uso de MAS y ficheros de 150 MB

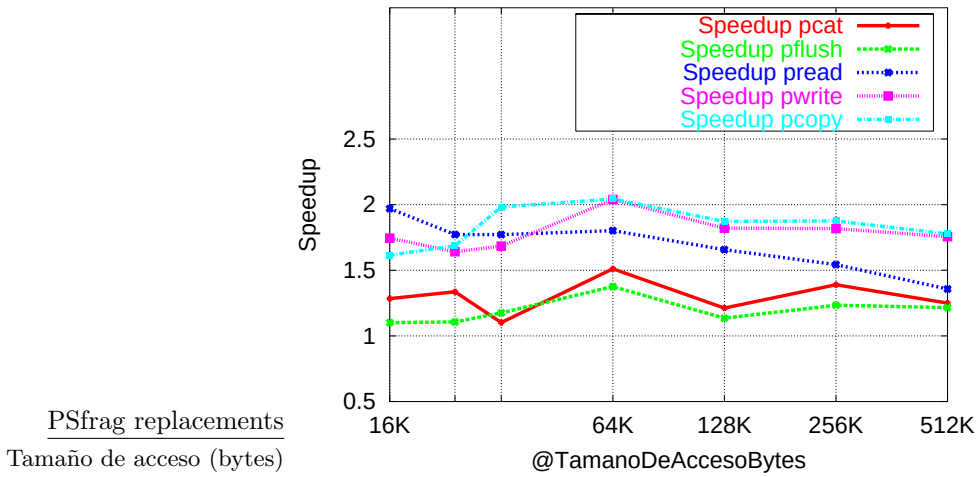


Figura 11.16: Speedup de las operaciones de E/S

11.7. Evaluación de operaciones que explotan el paralelismo en el cliente

Hasta este punto se ha realizado una evaluación de soluciones que sólo explotan el paralelismo en el servidor. No obstante, las aplicaciones que en mayor medida permiten beneficiarse de un sistema de ficheros paralelo y, en particular, del sistema de ficheros MAFPS son aquéllas que explotan el paralelismo en la parte cliente.

Para llevar a cabo esta evaluación, se ha implementado una aplicación que permite realizar la multiplicación de dos matrices almacenadas en fichero y cuyo resultado se escribirá también en fichero. Se han implementado dos diferentes versiones de esta aplicación, una paralela y otra secuencial:

- La aplicación paralela (`pmatrix`) se encarga de realizar accesos a los ficheros que almacenan las matrices, procesando diferentes filas de la matriz en paralelo. Debido a que como entorno de ejecución se utiliza un grupo de almacenamiento formado por 4 nodos, el número de filas

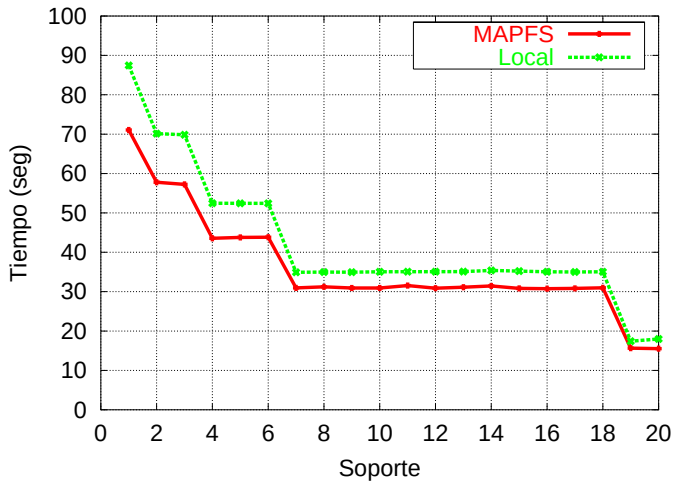


Figura 11.17: Evaluación de la ejecución del algoritmo Apriori con MAPFS y en local

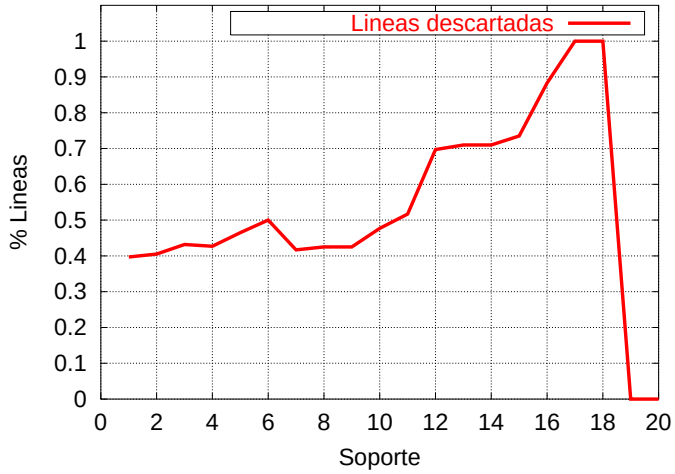


Figura 11.18: Porcentaje de líneas descartadas en Apriori con uso de hints

procesadas de forma paralela es 4 y en cada iteración, la aplicación crea 4 clientes que acceden simultáneamente a los ficheros.

- La aplicación secuencial (`smatrix`) realiza el proceso de una forma secuencial y accediendo a los datos en local.

En la figura 11.20 se recoge la evaluación de la ejecución de ambas aplicaciones, utilizando un fichero de matrices de 100 MB y diferentes tamaños de acceso, desde 1 KB a 8 KB.

Si se utiliza la multiplicación de matrices en conjunción con el MAS proxy, los resultados son aún mejores. Como se puede comprobar en la figura 11.21, el *speedup* de esta solución está por encima de 3,7 y a medida que se incrementa el tamaño de acceso se aproxima a 4, que es el valor máximo teórico[2], obteniendo un valor de 3,98 para un tamaño de acceso de 8 KB.

[2]salvo en casos de *speedup superlineal*

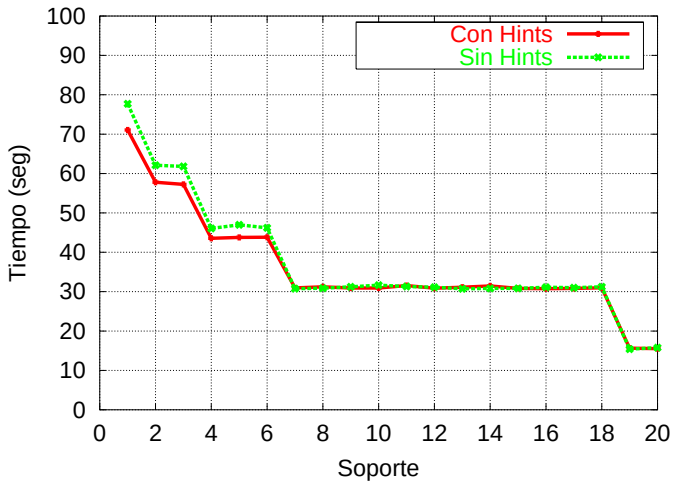
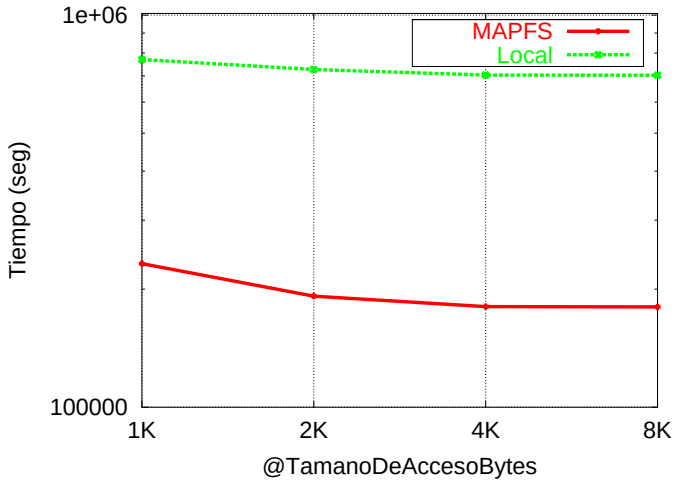


Figura 11.19: Evaluación de la ejecución del algoritmo Apriori con y sin hints



PSfrag replacements
Tamaño de acceso (bytes)

Figura 11.20: Evaluación de la ejecución de la multiplicación de matrices

Otro de los aspectos que hay que tener en cuenta a la hora de evaluar la arquitectura MAPFS es el comportamiento del sistema respecto al número de procesos que ejecutan una determinada aplicación. La tabla 11.1 muestra las diferencias de tiempo de proceso por fila de la aplicación `pmatrix`, dependiendo del número de procesos utilizados. Como se puede observar, la diferencia no es significativa, debido a que para este tipo de aplicaciones, la solución óptima es aquélla cuyo número de procesos es igual al número de nodos del grupo de almacenamiento correspondiente.

11.8. Evaluación de los grupos de almacenamiento

De cara a evaluar el funcionamiento del sistema MAPFS en una situación real, es necesario validar el uso de los grupos de almacenamiento en un escenario determinado. Para llevar a cabo esta validación, se han utilizado los dos clusters descritos anteriormente y se han definido dos grupos de almacenamiento

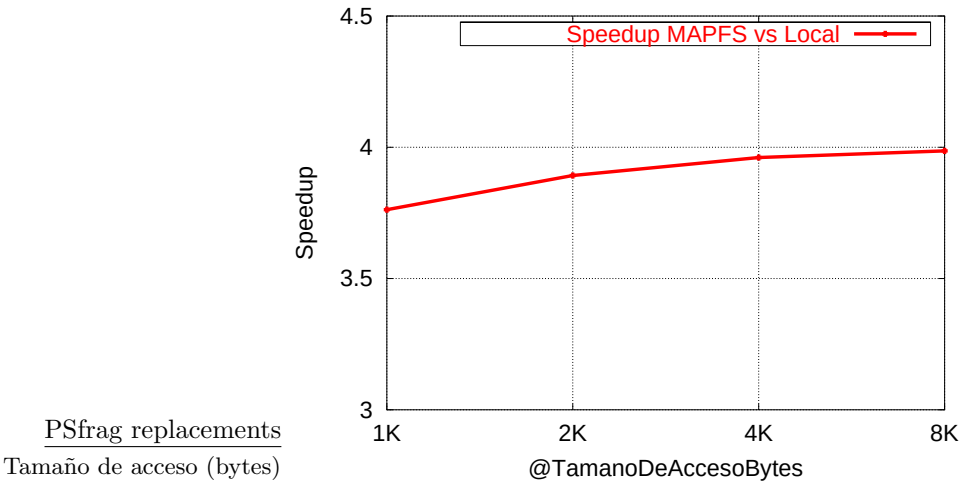


Figura 11.21: Speedup de la solución paralela de multiplicación de matrices

Tamaño de bloque (bytes)	Número de procesos	Tiempo (segundos)
1024	4	2,2690
1024	8	2,2296
1024	16	2,2422
2048	4	1,8745
2048	8	1,8765
2048	16	1,8777
4096	4	1,7631
4096	8	1,7611
4096	16	1,7598
8192	4	1,7598
8192	8	1,7540
8192	16	1,7561

Cuadro 11.1: Tiempos de proceso de la aplicación `pmatrix` dependiendo del número de procesos

siguiendo la política de agrupación por similitud, debido a las diferencias técnicas de los servidores que componen ambos clusters. El primer grupo de almacenamiento, que denominaremos G_1, está formado por los cuatro servidores pertenecientes al primer cluster. El segundo grupo de almacenamiento, G_2, está compuesto por seis de los ocho servidores del segundo cluster. Por tanto, el escenario inicial se puede describir de la siguiente forma:

$$G_1 = \{S_{1(1)}, S_{1(2)}, S_{1(3)}, S_{1(4)}\}$$
$$G_2 = \{S_{2(1)}, S_{2(2)}, S_{2(3)}, S_{2(4)}, S_{2(5)}, S_{2(6)}\}$$

Estos dos grupos de almacenamiento se utilizan durante un determinado intervalo de tiempo. Las operaciones de E/S se pueden realizar sobre cualquiera de los dos grupos de almacenamiento, siguiendo como criterio de elección las necesidades correspondientes a la aplicación que utiliza dichas operaciones

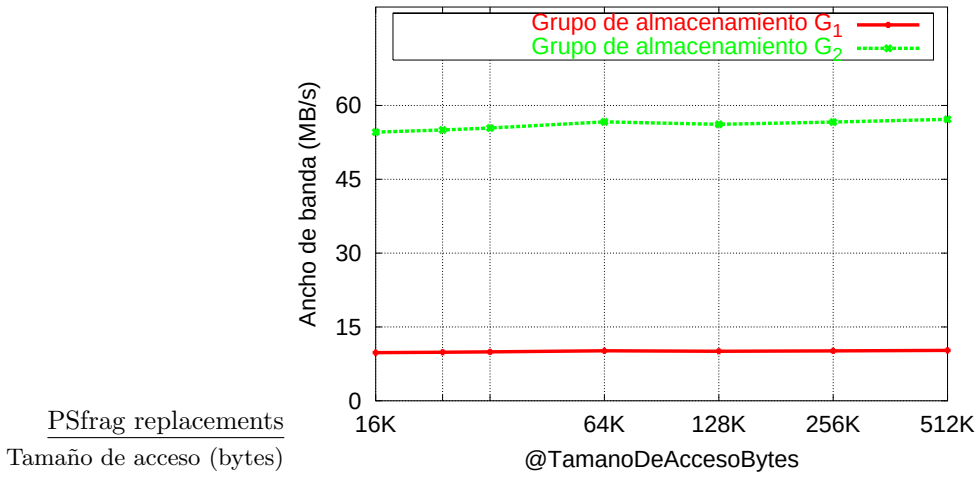


Figura 11.22: Ancho de banda proporcionado por los grupos de almacenamiento G_1 y G_2 en una operación de escritura de un fichero de 150 MB

de E/S. El grupo de almacenamiento G_2 proporciona mejor rendimiento. Por otro lado, el grupo de almacenamiento G_1 ofrece mayor capacidad de almacenamiento. La figura 11.22 muestra los anchos de banda proporcionados por ambos grupos para una operación de escritura de un fichero de 150 MB.

Dependiendo de la demanda de la aplicación y la carga de los clusters, así como del espacio de almacenamiento requerido, será conveniente utilizar uno de los dos grupos. Supongamos que en los grupos de almacenamiento G_1 se han almacenado ficheros que ocupan 1 GB y en el grupo de almacenamiento G_2 ficheros que ocupan 2 GB.

A continuación, se realiza un cambio en la topología del sistema, añadiendo dos servidores al grupo de almacenamiento G_2, según se muestra en la figura 11.23.

Los dos servidores añadidos constituyen un nuevo grupo secundario que denominaremos G_3 y que suponemos que está vacío. Por otro lado, el grupo de almacenamiento G_2 pasa a ser otro grupo secundario. Finalmente, el grupo G'_4 está formado por los 8 servidores del segundo cluster. Los grupos secundarios se representan sombreados en la figura.

Al añadir 2 nodos al segundo grupo de almacenamiento principal, el rendimiento de las operaciones realizadas sobre este grupo se incrementan. La figura 11.24 representa los anchos de banda de ambos grupos en una operación de escritura.

Las aplicaciones sólo ven los grupos principales. No obstante, las operaciones realizadas sobre ficheros del grupo G'_4 que se hubieran creado antes de añadir los dos servidores, el sistema las proyecta sobre el grupo G_2 de forma transparente al usuario. Las operaciones con estos ficheros seguirán proporcionando un ancho de banda similar al representado en la figura 11.22.

Vamos a suponer que después de llevar a cabo la adición de los servidores, al finalizar las operaciones de E/S realizadas en el sistema, el grupo G'_1 contiene ficheros que ocupan 4 GB y el grupo G'_4 ficheros que ocupan 12 GB, distribuyéndose estos últimos ficheros entre el grupo secundario G_2 (10 GB) y el grupo principal G'_4 (2 GB).

Si se realiza una operación de defragmentación, se lleva a cabo una reconstrucción de los datos de los grupos secundarios sobre los grupos principales a los que pertenecen. En este caso, sólo será necesario reconstruir los datos del grupo secundario G_2, debido a que el grupo G_3 está vacío. Suponiendo que el sistema ha utilizado un tamaño de acceso de 16K, el tiempo necesario para realizar esta reconstrucción

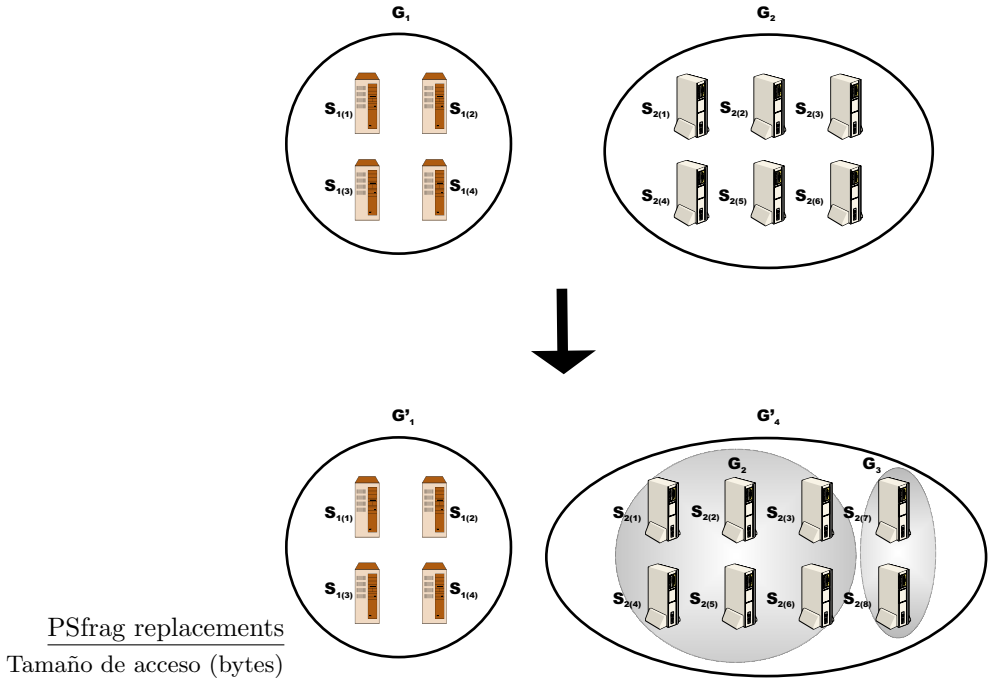


Figura 11.23: Cambio de topología en el sistema

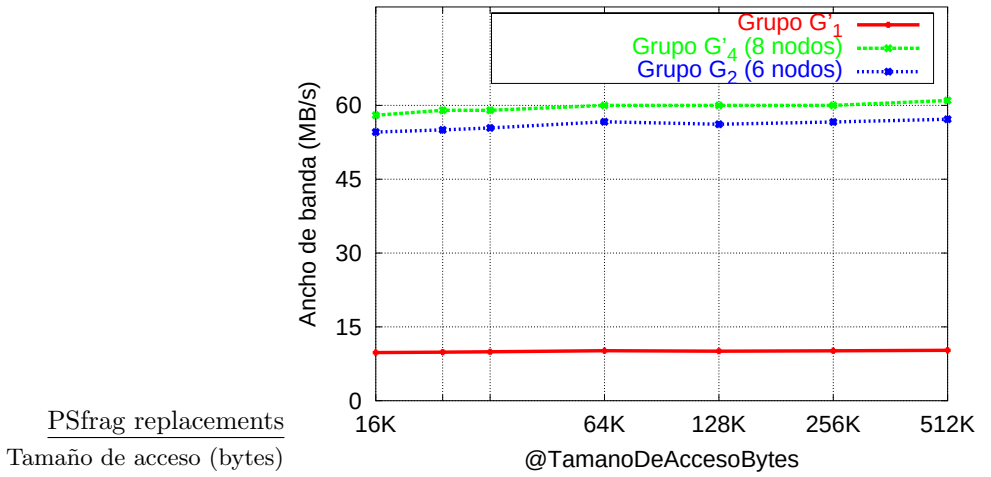


Figura 11.24: Ancho de banda proporcionado por los grupos de almacenamiento G'_1 y G'_4 en una operación de escritura de un fichero de 150 MB

es de aproximadamente 6 minutos, que corresponde al tiempo de lectura de los 10 GB contenidos en el grupo de almacenamiento secundario y volcado al grupo de almacenamiento principal asociado. Esta operación es conveniente realizarla durante períodos de baja carga del sistema, para evitar entrar en conflicto con las operaciones de E/S ordinarias.

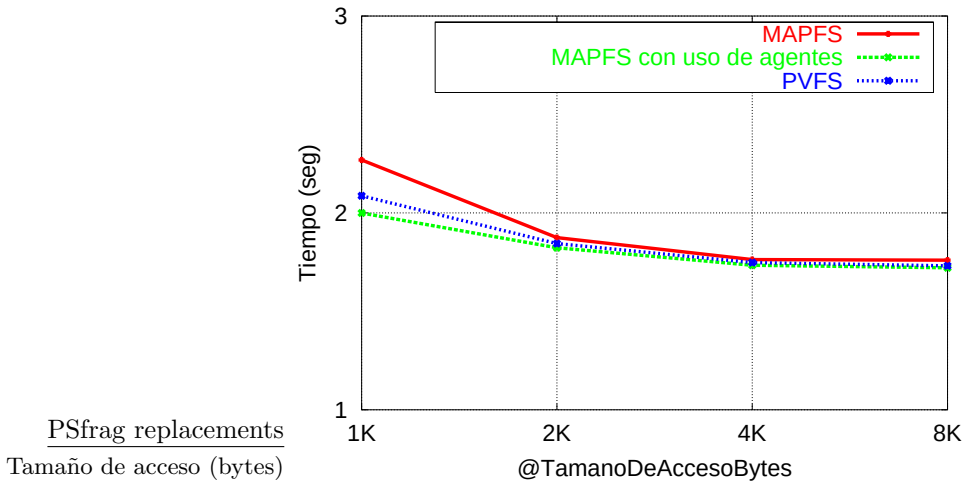


Figura 11.25: Comparación de la ejecución de la multiplicación de matrices de PVFS frente a MAPFS (una única fila)

11.9. Comparación con PVFS

Este capítulo no sería completo si la solución propuesta no fuera comparada con otro sistema de ficheros paralelo. Para realizar dicha comparación, se ha elegido PVFS, por tener como principales ventajas las siguientes [CIRT00]:

- Se trata de un sistema de ficheros paralelo desarrollado para *clusters* Linux.
- Proporciona un gran ancho de banda en operaciones de lectura y escritura concurrentes realizadas desde múltiples procesos o threads a un fichero común.
- Es robusto y escalable.
- Da soporte a múltiples APIs: un API nativa de PVFS, la interfaz de E/S UNIX/POSIX así como otras APIs, entre las que se incluye MPI-IO.
- Es de libre distribución y sencillo de instalar.
- Permite beneficiarse tanto de paralelismo en el cliente como en el servidor.

Para realizar la comparación, se ha implementado la multiplicación de matrices en PVFS, a través de su interfaz nativa.

[htb]

La figura 11.25 representa el tiempo de ejecución de las solución PVFS y de la solución MAPFS para una única fila de la matriz, con o sin uso de MAS. El MAS se encarga de realizar el *prefetching* de las filas y columnas de las matrices implicadas, según se describe en la sección 8.4.

Los resultados muestran que en el caso del procesamiento de una única fila:

- La solución MAPFS sin uso de MAS es ligeramente más lenta que la solución PVFS.
- La solución MAPFS con uso de MAS es sin embargo más rápida que la solución PVFS.
- A medida que se incrementa el tamaño de los ficheros de entrada, todas las soluciones convergen en velocidad.

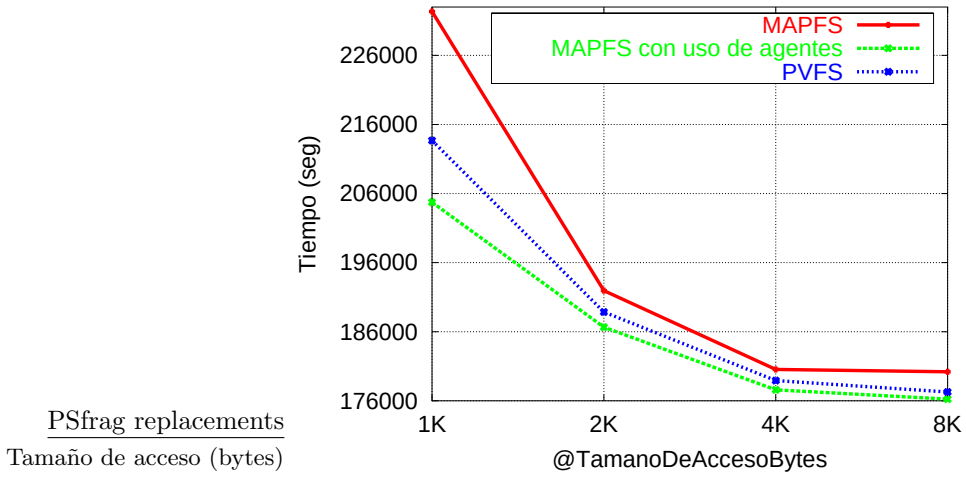


Figura 11.26: Comparación de la ejecución de la multiplicación de matrices de PVFS frente a MAPFS (fichero completo de 100 MB)

- En todo caso y debido al hecho de que PVFS ofrece un gran rendimiento, ambas soluciones MAPFS son satisfactorias, llegando incluso a mejorar la solución dada por PVFS.

No obstante, a medida que se incrementa el número de filas procesadas, las diferencias en los tiempos de ejecución empiezan a ser significativas. En la figura 11.26 se muestra la evaluación de la ejecución de la multiplicación de un fichero completo de matrices de tamaño 100 MB en PVFS y en MAPFS, con y sin MAS.

11.10. Resumen

Este capítulo recoge una evaluación de diferentes tipos de aplicaciones que utilizan el sistema de ficheros MAPFS y que se benefician de diferentes aspectos del mismo:

- Paralelismo en el servidor, que es inherente al propio sistema. Se ha comprobado que con determinados tamaños de acceso y simplemente haciendo uso de esta característica, las aplicaciones pueden mejorar sustancialmente el acceso a la información.

- Paralelismo en el cliente: Si ejecutamos varios clientes que acceden en paralelo a la información, el *speedup* del sistema se acerca al número de nodos o número de clientes, obteniendo un valor próximo al *speedup* máximo.

- Uso del sistema multiagente: El sistema de ficheros MAPFS permite el uso de un subsistema multiagente que le proporciona determinadas características adicionales. En las pruebas realizadas, se ha evaluado el rendimiento de las operaciones de E/S haciendo uso de un MAS que tiene funcionalidad de *proxy*; concretamente, se ha evaluado la operación de lectura con *prefetching*.

- Uso de *hints*: Otra ventaja del sistema de ficheros MAPFS es la posibilidad de que la aplicación utilice estructuras de control que se proyectan en determinados *hints*, los cuáles informan sobre la semántica de los datos. En este apartado se han llevado a cabo dos evaluaciones diferentes, una evaluación de la implementación del algoritmo Apriori con creación de *hints* de forma dinámica

por parte del MAS asociado y la evaluación de una multiplicación de matrices, a través del uso de estructuras de control de usuario, proporcionadas por la propia aplicación. En ambos casos, el uso de estas estructuras permite mejorar el rendimiento.

Finalmente, se ha realizado una comparación del sistema de ficheros MAPFS frente a PVFS, observando que ambas soluciones son similares, llegando incluso a ser mejor la primera, cuando utiliza un MAS. Por tanto, estas evaluaciones validan de forma cuantitativa la solución ofrecida por MAPFS.

Parte III

CONCLUSIONES Y LÍNEAS FUTURAS

Capítulo 12

Conclusiones y líneas de trabajo futuras

12.1. Introducción

La realización de la presente tesis ha alcanzado de una forma adecuada los objetivos fijados. La consecución de estas metas se ha materializado en el diseño e implementación de la arquitectura de E/S MAPFS. Este trabajo tiene como **principal aportación el uso de sistemas multiagente en arquitecturas de E/S**.

Otras aportaciones del trabajo han supuesto la definición de un marco completo de trabajo en el que las aplicaciones pueden especificar sus demandas de E/S y adaptarse a la arquitectura de E/S subyacente. Esta característica le proporciona un valor añadido al sistema MAPFS, debido a la idea de que el sistema de E/S no debe ser un elemento aislado y totalmente transparente a las aplicaciones, sino que debe integrarse dentro del sistema global como un elemento más, principalmente por el hecho de que constituye un importante cuello de botella.

A continuación se analizan más en detalle las aportaciones realizadas, que satisfacen las hipótesis planteadas en el capítulo 1 .

12.2. Aportaciones en el campo de los sistemas de ficheros

La construcción del sistema de ficheros MAPFS ha permitido fundir dos áreas que parecen tener bastante poco en común: la E/S paralela y la tecnología de agentes. Para lograr dicho objetivo, se han utilizado algunas tecnologías que tienden puentes entre ambos campos, a saber:

- Uso de mecanismos de comunicación, tales como RPC o MPI.
- Uso de mecanismos de concurrencia y paralelismo, tales como MPI.
- Abstracciones y conceptos procedentes del campo de la agencia, tales como sistemas multiagente, cooperación entre agentes o *performatives* KQML.

La distancia, a veces insalvable, que existe entre la filosofía de construcción de sistemas de bajo nivel y el uso de la tecnología de agentes se ha solventado en su mayor parte separando las cuestiones conceptuales y de diseño de las cuestiones de implementación y uso de herramientas de desarrollo, de tal forma que finalmente se desechó el uso de herramientas propias de entornos de agentes por no adaptarse correctamente al problema planteado, sin desechar no obstante todas las posibilidades conceptuales que proporcionan dichos entornos. De este modo, el desarrollo de MAPFS se ha beneficiado de las ventajas del uso de la teoría de agentes, demostrando de este modo que **es factible utilizar paradigmas de alto nivel en la implementación de sistemas, tales como la construcción de un sistema de ficheros paralelo**. Algunas de estas ventajas son:

- Dotar a los procesos pertenecientes al sistema MAPFS_MAS de características propias de los agentes, tales como la autonomía, la proactividad y la reactividad.
- Dotar al sistema de un modelo de coordinación que permite gestionar las operaciones de todos los procesos involucrados.
- Incrementar la eficiencia de la solución obtenida a través principalmente del modelo de coordinación utilizado.
- Aumentar las funcionalidades del sistema de una forma modular y flexible.

De una manera global, la construcción de MAPFS ha permitido el desarrollo de un sistema de ficheros paralelo en tres niveles totalmente diferentes, a saber:

1. Se ha realizado un desarrollo completo de un sistema de ficheros a nivel de cliente.
2. A nivel de servidor, se ha permitido configurar los grupos de almacenamiento, que permiten abstraer el papel del servidor y llevar a cabo una reconfiguración dinámica del sistema, en base a determinados parámetros o características del mismo.
3. A nivel de aplicación, se han definido los perfiles de E/S, que permiten la adaptación del sistema de almacenamiento a las necesidades de cada aplicación.

Este último nivel aporta a la solución capacidad para incrementar el rendimiento de las aplicaciones. Por este motivo, el sistema MAPFS es especialmente apropiado para su uso por parte de aplicaciones HPC.

12.3. Aportaciones en el campo de las aplicaciones de alto rendimiento

Tradicionalmente el sistema de E/S ha sido marginado a favor de la creación de procesadores cada vez más potentes. No obstante, la importancia adquirida por el acceso y tratamiento de la información en las nuevas aplicaciones ha supuesto un auge en esta disciplina. Este tipo de aplicaciones requieren accesos a grandes conjuntos de datos a grandes velocidades.

Por otro lado, también ha existido una tendencia en E/S a ofrecer interfaces y comportamientos homogéneos independientemente de las aplicaciones que los utilizan. Esto conlleva un decremento del rendimiento de algunas de las aplicaciones, que pueden ejecutarse de una forma más eficiente si proporcionan al sistema patrones de acceso a los datos.

En este sentido, esta tesis **define los perfiles de E/S, como un conjunto de estructuras que permiten especificar la demanda de E/S de una determinada aplicación**. Los perfiles incluyen la información correspondiente a la topología del sistema, al MAS que da soporte al sistema

de ficheros y a las estructuras de control de usuario, que permiten a las aplicaciones informar sobre los patrones de acceso a los datos. Estas estructuras de control se traducen a *hints*, a fin de incrementar el rendimiento de las operaciones de E/S. No obstante, también es posible que el MAS asociado cree *hints* de forma dinámica durante la ejecución de la aplicación.

En el plano de las evaluaciones, **se ha realizado una evaluación exhaustiva** de varias aplicaciones que utilizan el sistema MAPFS de diferentes formas, **concluyéndose que el uso de este sistema mejora el rendimiento de las mismas**. De forma significativa, la evaluación de una multiplicación de matrices ha permitido validar el uso de estructuras de control de usuario y la evaluación del algoritmo Apriori, como aplicación dentro del campo de *Data Mining*, ha supuesto la validación de la creación de *hints* por parte del MAS.

Las aportaciones realizadas por este trabajo pueden ser clasificadas como teóricas, cuando ofrecen marcos, conceptos o formalismos aplicables a escenarios genéricos, o prácticas, si corresponden a diseños o implementaciones concretos de las anteriores soluciones. A continuación se detallan las aportaciones de la presente tesis desde estos dos puntos de vista.

12.4. Aportaciones teóricas

Algunas de las aportaciones teóricas de esta tesis proceden del uso de la teoría de agentes a un campo tradicionalmente ajeno al mismo, como es el de la E/S. En este terreno, se ha definido **una arquitectura multiagente lo suficientemente general** como para permitir que sea utilizada de una forma eficiente por aplicaciones tan diferentes entre sí como son las aplicaciones científicas o las aplicaciones de *Data Mining*. Este trabajo da suficientes referencias de la adaptación de diferentes dominios a la misma. En este aspecto, la inclusión de los perfiles de E/S toma un papel destacado al constituir el puente entre la aplicación y la arquitectura de E/S. Dicha arquitectura genérica también facilita la inclusión de funcionalidades adicionales.

La otra gran aportación teórica de este trabajo es la **definición de los grupos de almacenamiento**, que consisten en un conjunto de formalismos matemáticos que permiten modelar y gestionar el sistema de almacenamiento de una forma óptima. Los grupos de almacenamiento pretenden ser una abstracción lógica del concepto de servidor, haciendo posible su gestión dinámica. La definición de parámetros correspondientes a los grupos de almacenamiento permite establecer de una forma analítica un conjunto de políticas de creación de los mismos, que sean aplicables a diferentes escenarios de computación.

12.5. Aportaciones prácticas

Respecto a las aportaciones prácticas, hay que destacar que se ha realizado el **diseño completo de una arquitectura multiagente de E/S paralela para clusters**, así como la **implementación de un primer prototipo de esta arquitectura**, en la que se utiliza NFS como sistema de ficheros en el servidor. Asimismo, se ha definido una **interfaz de E/S** con un amplio abanico de operaciones y una **interfaz genérica y flexible que permite que las aplicaciones describan patrones de acceso**, a fin de optimizar sus accesos al sistema de E/S.

La evaluación de la implementación realizada ha sido muy satisfactoria, mejorando sistemas de E/S existentes y constituyendo una nueva manera de abordar la E/S.

12.6. Líneas de trabajo futuro

Según hemos analizado en este capítulo, los objetivos que se plantean al comienzo de la tesis se han alcanzado de una forma satisfactoria. Además, esta tesis abre nuevas líneas de trabajo, que permiten extender las metas logradas. Estas líneas están orientadas tanto a incrementar la funcionalidad de la solución propuesta como a desarrollar nuevos aspectos relacionados con la misma. En ambos casos, se puede distinguir entre líneas de desarrollo o implementación y líneas de investigación.

12.6.1. Líneas de desarrollo

Respecto al desarrollo del sistema propuesto, hay que indicar que se ha realizado un primer prototipo que utiliza NFS. Esta primera versión es limitada, ya que sólo incluye la implementación de las operaciones de E/S básicas. Una primera tarea consiste en la extensión de estas operaciones, llevando a cabo una implementación de operaciones avanzadas, colectivas y misceláneas, que permitan incrementar la funcionalidad proporcionada por el sistema e incluso mejorar el rendimiento del mismo con el uso de operaciones más adaptadas a un entorno paralelo.

Otra línea de trabajo relacionada consiste en el desarrollo del sistema MAPFS utilizando otros tipos de servidores de ficheros e integrándolos con el prototipo ya construido. De este modo, se permite el uso de sistemas heterogéneos en la parte servidora de la arquitectura, mediante el uso de una interfaz unificada.

Además, el prototipo de MAPFS realizado no proporciona tolerancia a fallos, no habiéndose implementado un agente de tolerancia a fallos. Éste y otros servicios adicionales constituyen líneas de desarrollo futuros. De hecho, el agente cache considerado no utiliza ningún protocolo de coherencia, ya que este aspecto queda fuera de los objetivos de la tesis. Otra posible línea es proporcionar esta característica al agente cache.

Respecto a la escritura de ficheros, es necesario plantear una semántica de coutilización, que permita especificar el efecto de varios procesos accediendo de forma simultánea a un mismo fichero MAPFS. Será, por tanto, necesario, ampliar la interfaz con operaciones que permitan establecer cerrojos sobre dichos ficheros.

En el campo de los grupos de almacenamiento, se pretente realizar como trabajo futuro la adición de nuevas funciones de distribución y su posterior evaluación. Estas funciones de distribución se realimentarán con la información recogida por los parámetros de agrupación de MAPFS.

12.6.2. Líneas de investigación

Además de las líneas futuras de desarrollo, con la realización de esta tesis se abren nuevas líneas de investigación, que permiten el desarrollo de nuevas tesis. Dada la factibilidad de desarrollo de un sistema de ficheros utilizando la teoría de agentes, como trabajo futuro se puede plantear la construcción de diferentes gestores del sistema operativo, siguiendo la misma filosofía. Además esta alternativa queda respaldada por diferentes trabajos realizados, ya que los agentes han sido utilizados de forma extensiva como gestores de recursos [GMM+01], [MKKS01], característica básica de un sistema operativo. De hecho, estos trabajos utilizan el concepto de planificador y de *dispatcher* de tareas. En [GF01] incluso se habla de *micro-kernel* de agentes como núcleo sobre el que descansan una serie de módulos de sistema y módulos de aplicación, que actúan como un conjunto de servicios o facilidades que permiten el desarrollo eficiente de herramientas para agentes.

Por otro lado, dadas las características del sistema multiagente, sería interesante utilizar algoritmos que permitan entrenar a los agentes, a fin de mejorar el conocimiento de los mismos y por tanto su funcionamiento. Por ejemplo, en el caso del agente cache sería posible el uso de algoritmos o heurísticas

que le permitan el cálculo óptimo de los factores de *caching* y *prefetching*.

Por último y debido a que MAPFS proporciona una interfaz de acceso a datos de alto rendimiento, es posible utilizarlo como infraestructura subyacente para determinadas aplicaciones grid. Una plataforma grid de altas prestaciones involucra un conjunto heterogéneo de nodos de computación que pueden residir en diferentes ubicaciones, poseer una estructura diferente y utilizar diferentes políticas. El uso de MAPFS como plataforma grid para aplicaciones de datos intensivas ya se ha propuesto en [PCG+03b]. La flexibilidad del sistema de ficheros MAPFS y el uso de grupos de almacenamiento le convierten en una opción muy interesante como núcleo de una plataforma grid que proporcione servicios para el acceso eficiente a datos. MAPFS permite extender este tipo de servicios de forma incremental.

Apéndice A

Operaciones de E/S del sistema de ficheros MAPFS

A.1. Introducción

La mayoría de los sistemas de ficheros paralelos existentes utilizan una interfaz similar a la del sistema de ficheros UNIX, limitándose a operaciones de apertura, cierre, lectura, escritura y posicionamiento de ficheros y sin operaciones más avanzadas que permitan beneficiarse de la característica de paralelismo.

No obstante, MAPFS ofrece una interfaz que permite realizar operaciones más eficientes en un entorno paralelo.

A continuación se describen las operaciones del sistema de ficheros MAPFS clasificadas por su tipo, mostrando su estructura y funcionalidad.

A.2. Operaciones básicas

Son las operaciones típicas de un sistema de ficheros, que permiten llevar a cabo tareas básicas del mismo.

1. Operación de apertura:

   ```
   Mapfs_File mapOpen (Mapfs_Comm comm, char *global_filename, int access_mode,
   int flags, Mapfs_Hints *hints, int perm, int *error_code);
   ```

   Los parámetros que recibe esta función tienen la siguiente lectura:

   - `comm`: todas las aperturas se consideran operaciones colectivas (véase más adelante operaciones de E/S colectivas). El comunicador *comm* especifica el proceso participante.
   - `global_filename`: especifica el nombre global del fichero.

- `access_mode`: especifica el modo de acceso del fichero, (`MAP_CREATE`, `MAP_RDONLY`, `MAP_WRONLY`, `MAP_RDWR`, `MAP_TRUNC`, `MAP_DELETE_ON_CLOSE`, `MAP_EXCLUSIVE` o `MAP_ATOMIC`).
- `flags`: En el caso de creación del fichero (`access_mode == MAP_CREATE`), se debe especificar además el tipo de modelo de distribución de los datos del mismo. En caso contrario, el fichero ya tendrá establecida dicha característica.
- `hints`: puede utilizarse para pasar "pistas" *hints* a la implementación del sistema de ficheros MAPFS para una posible mejora del rendimiento. Ejemplos de *hints* incluyen la especificación de la disposición del disco, información sobre *prefetching/caching*, estilos de acceso a ficheros, patrones de particionado de datos e información requerida para uso en sistemas heterogéneos. Los *hints* son opcionales. Si no están definidos, se utilizan valores por defecto. Su formato se describe en la sección A.6.
- `perm`: Especifica los permisos de acceso del fichero al estilo UNIX.
- `error_code`: El éxito o fallo de la operación de apertura se recoge en este parámetro.

La operación devuelve un valor del tipo *Mapfs_File*, que actuará como descriptor de fichero.

2. Operación de cierre:

```
void mapClose (Mapfs_File fd, int *error_code)
```

La operación de cierre también es colectiva. Todos los procesos que abrieron el fichero, deben cerrarlo. La operación recibe la estructura *fd* que corresponde al descriptor del fichero devuelto por la operación mapOpen y devuelve un código de error correspondiente a la finalización de la misma.

3. Operación de creación:

```
Mapfs_File mapCreat (Mapfs_Comm comm, char *global_filename,
Mapfs_Hints *hints, int perm, int *error_code);
```

Los parámetros que recibe esta función son equivalentes a los de la operación `mapOpen`. De hecho, la llamada

```
mapCreat(comm, global_filename, hints, perm, &error_code);
```

es equivalente a

```
mapOpen (comm, global_filename, MAP_CREATE|MAP_TRUNC|MAP_WRONLY, hints, perm,
&error_code);
```

4. Operaciones de lectura y escritura contigua:

```
void mapReadContig (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Status *status, int *num_bytes)

void mapWriteContig (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Status *status, int *num_bytes)
```

Las rutinas de lectura/escritura contiguas se utilizan cuando los datos a leer o escribir están contiguos tanto en memoria como en el fichero. `mapReadContig` y `mapWriteContig` son versiones independientes y bloqueantes de las llamadas `read` y `write`. Son las operaciones más sencillas de lectura y escritura que ofrece la interfaz del sistema de ficheros MAPFS. Los parámetros que reciben estas funciones son:

- `fd`: corresponde al descriptor del fichero sobre el que se va a hacer la operación de lectura o escritura.
- `buf`: se trata de la dirección del *buffer* en memoria en el cuál se almacenarán los bytes leídos del fichero o donde se encuentran los bytes que se van a escribir en el fichero.
- `offset:` desplazamiento a partir del inicio desde el cuál se va a realizar la lectura o escritura.
- `size`: número de bytes de datos que deben leerse o escribirse en el fichero.
- `status`: recoge información sobre la operación realizada, tal como el éxito o fallo de la operación de lectura o escritura.
- `num_bytes`: recoge información sobre la cantidad de datos realmente leídos o escritos.

5. Operación de modificación de puntero:

```
Mapfs_Offset mapSeek (Mapfs_File fd, Mapfs_Offset offset, int whence,
int *error_code)
```

Esta función puede utilizarse para cambiar la posición del puntero de fichero. Los parámetros que recibe esta función son:

- `fd`: corresponde al descriptor del fichero sobre el que se va a realizar la operación de posicionamiento del puntero.
- `offset`: desplazamiento respecto al punto de referencia sobre el cual se posicionara el puntero.
- `whence`: puede tomar los valores `MAP_SEEK_SET`, `MAP_SEEK_CUR` o `MAP_SEEK_END`, de forma análoga a la llamada al sistema de UNIX `lseek`.
- `error_code`: El éxito o fallo de la operación de posicionamiento del puntero se recoge en este parámetro.

Esta operación devuelve el valor del puntero resultante.

6. Operación para establecer u obtener información sobre un fichero abierto:

```
void mapFcntl (Mapfs_File fd, int cmd, Mapfs_Fcntl_t *fcntl,
int *error_code)
```

Se puede utilizar para establecer u obtener información sobre un fichero abierto, tal como el desplazamiento, modo de acceso y "pistas" (*hints*).

Los parámetros que recibe esta función son:

- `fd`: corresponde al descriptor del fichero sobre el que se va a realizar la operación.
- `cmd`: comando que se utilizará para aplicarle al fichero abierto correspondiente.
- `fcntl`: estructura que contiene los valores a establecer o recuperar respecto al fichero.
- `error_code`: El éxito o fallo de la operación se recoge en este parámetro.

Es similar al establecimiento de vistas en el caso de ROMIO [TGL99a].

A.3. Operaciones avanzadas

Se trata de operaciones que presentan los sistemas de ficheros paralelos incrementando la funcionalidad de su interfaz y mejora de rendimiento de las aplicaciones.

1. Operaciones de lectura y escritura no contigua:

```
void mapReadStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Status *status, int *error_code)

void mapWriteStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Status *status, int *error_code)
```

   Las operaciones `mapReadStrided` y `mapWriteStrided` son versiones bloqueantes de las llamadas *read* y *write* que soportan cualquier tipo de acceso no contiguo que pueda expresarse mediante los parámetros que reciben.

   Los parámetros recibidos por estas funciones son:

   - `fd`: corresponde al descriptor del fichero sobre el que se va a realizar la operación de E/S no contigua.
   - `buf`: se trata de la dirección del *buffer* en memoria en el cuál se almacenarán los bytes leídos del fichero o donde se encuentran los bytes que se van a escribir en el fichero.
   - `mem_list_count`: es el número de entradas en `mem_offsets` y `mem_lengths`.
   - `mem_offsets` y `mem_lengths`: son listas de desplazamientos y longitudes que permiten representar localizaciones de memoria no contiguas.
   - `file_list_count`: es el número de entradas en `file_offsets` y `file_lengths`.
   - `file_offsets` y `file_lengths`: son listas de desplazamientos y longitudes que permiten representar localizaciones en el fichero no contiguas.
   - `status`: recoge información sobre la operación realizada, tal como la cantidad de datos realmente leídos o escritos.
   - `error_code`: El éxito o fallo de la operación de lectura o escritura se recoge en este parámetro.

   Esta interfaz se puede considerar como una generalización del *readv/writev* de UNIX permitiendo la no contigüidad en el fichero.

   Se debe observar que `mapReadContig` y `mapWriteContig` son casos especiales de `mapReadStrided` y `mapWriteStrided`.

2. Operaciones de lectura y escritura no bloqueantes: La interfaz del sistema de ficheros MAPFS proporciona versiones no bloqueantes de todas sus llamadas de lectura y escritura.

```
void mapIreadContig(Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes)
```

```
void mapIwriteContig (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes)

void mapIreadStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code)

void mapIwriteStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code)

void mapIreadContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes)

void mapIwriteContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes)

void mapIreadStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code)

void mapIwriteStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code)
```

Las rutinas no bloqueantes devuelven un objeto `request` que se puede utilizar para comprobar la finalización de la operación. Para ello, se utilizan las siguientes rutinas:

```
int mapReadDone (Mapfs_Request *request)

int mapWriteDone (Mapfs_Request *request)
```

Estas rutinas hacen una comprobación sobre el descriptor `request` para determinar si la operación se ha completado y no requiere una acción posterior. Devuelven *true* si la operación ha sido completa y *false* en caso contrario.

```
int mapReadIcomplete (Mapffs_Request *request, Mapfs_Status *status,
int *error_code)

int mapWriteIcomplete (Mapfs_Request *request, Mapfs_Status *status,
int *error_code)
```

Estas rutinas no se bloquean esperando que la operación se complete. Realizan algún proceso adicional necesario para completar la operación. Si la operación se ha completado, devuelve *true* y establece el valor de la variable `status`; en caso contrario, devuelve *false*. Si se ha detectado un error, devuelven *true* y establecen el valor de `error_code` de forma apropiada.

```
void mapReadComplete (Mapfs_Request *request, Mapfs_Status *status,
int *error_code)

void mapWriteComplete (Mapfs_Request *request, Mapfs_Status *status,
int *error_code)
```

Estas rutinas se bloquean hasta que la operación especificada se completa y se establece el valor de la variable `status`. Si se detecta un error, se actualiza el valor de `error_code` de forma apropiada y la función devuelve el control de ejecución.

Para operaciones no bloqueantes, se establece un mecanismo de *callback*, de forma que se pueda especificar una operación que se lleve a cabo cuando una operación no bloqueante haya finalizado. La interfaz del sistema de ficheros MAPFS proporciona versiones con posibilidad de establecer *callbacks* para todas las operaciones no bloqueantes de lectura y escritura. Las cabeceras de dichas funciones son:

```
void mapCalReadContig(Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes, void (*operation)())

void mapCalWriteContig (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes, void (*operation)())

void mapCalReadStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code, void (*operation)())

void mapCalWriteStrided (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code, void (*operation)())

void mapCalReadContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes, void (*operation)())

void mapCalWriteContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Request *request, int *num_bytes, void (*operation)())

void mapCalReadStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code, void (*operation)())
```

```
void mapCalWriteStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Request *request, int *error_code, void (*operation)())
```

El parámetro `operation` especifica la operación que debe ejecutarse después de que la operación de lectura y escritura correspondiente haya finalizado.

A.4. Operaciones colectivas

Para permitir el uso de E/S colectiva, MAPFS proporciona versiones colectivas de todas las rutinas de lectura y escritura, tanto operaciones contiguas como no contiguas.

```
void mapReadContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Status *status, int *num_bytes)

void mapWriteContigColl (Mapfs_File fd, char *buf, int offset, int size,
Mapfs_Status *status, int *num_bytes)

void mapReadStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Status *status, int *error_code)

void mapWriteStridedColl (Mapfs_File fd, char *buf, int mem_list_count,
long *mem_offsets, int *mem_lengths, int file_list_count,
Mapfs_Offset *file_offsets, int *file_lengths,
Mapfs_Status *status, int *error_code)
```

Los parámetros tienen el mismo significado que en las versiones básicas.

A.5. Operaciones misceláneas

Algunas operaciones adicionales para la gestión de ficheros son las siguientes:

1. Operación de eliminación de un fichero:

   ```
   void mapDelete (char *global_filename, int *error_code)
   ```

   Esta operación permite eliminar un fichero en el sistema MAPFS.

2. Operación de redimensionado de un fichero:

   ```
   void mapResize (Mapfs_File fd, Mapfs_Offset size, int *error_code)
   ```

   Esta operación sirve para redimensionar un fichero del sistema MAPFS.

3. Operación de renombrado de un fichero:

   ```
   void mapRename (char *old_filename, char *new_filename, int *error_code)
   ```

Esta operación permite modificar el nombre de un fichero, de *old_filename* a *new_filename*.

4. Operaciones de apertura y cierre de sesión:

```
void mapInit (int *argc, char ***argv, int *error_code)

void mapFinish (int *error_code)
```

Estas dos operaciones permiten abrir y cerrar una sesión para realizar gestiones con ficheros, inicializando algunos parámetros de configuración, así como ciertas estructuras de datos necesarias para dicha gestión.

A.6. Operaciones de configuración de *hints*

Son operaciones para establecer los *hints* del sistema y, por tanto, permiten modificar y acceder a los diferentes campos de los *hints*. Las operaciones de configuración de *hints* son las siguientes:

- `Mapfs_Hints * mapHintsNew(int block_ident)`: Esta operación crea una nueva estructura *hint* para el bloque cuyo identificador es `block_ident`.
- `void mapHintsFree(Mapfs_Hints *hint)`: Esta operación libera una estructura *hint*.
- `int mapHintsSet(Mapfs_Hints *hint, int code_field, void * value)`: Esta operación modifica un campo de la estructura *hint* con un valor. La operación devuelve 0 si se realiza de forma correcta y -1 en caso contrario.
- `void * mapHintsGet(Mapfs_Hints *hint, int code_field)`: Esta operación devuelve el valor de un campo concreto de la estructura *hint*. Si el campo no está definido, esta operación devuelve NULL.

A.7. Operaciones de control de usuario

Son las operaciones que utilizan de forma directa las aplicaciones de usuario. Las aplicaciones pueden gestionar el rendimiento del sistema sólo a través de este API.

Las operaciones de control de usuario son las siguientes:

- `Mapfs_CtrlUser * mapCtrlUserNew(Mapfs_Tuples *tupl)`: Esta operación crea una nueva estructura de control de usuario para un conjunto de tuplas representado por *tupl*.
- `void mapCtrlUserFree(Mapfs_CtrlUser *ctrlUser)`: Esta operación libera una estructura de control de usuario.
- `int mapCtrlUserSet(Mapfs_CtrlUser *ctrlUser, int code_field, void *expr)`: Esta operación modifica un campo de la estructura de control de usuario con una expresión. La operación devuelve 0 si se realiza de forma correcta y -1 en caso contrario.
- `void *mapCtrlUserGet(Mapfs_CtrlUser *ctrlUser, int code_field)`: Esta operación devuelve el valor de un campo concreto de una estructura de usuario de control. Si el campo no está definido, esta operación devuelve NULL.

Apéndice B

Teoría de conjuntos

B.1. Introducción

Una de las herramientas más básicas y útiles que proporciona la Ciencia Matemática es el concepto de **relación**. Este concepto permite el análisis y el estudio de los **conjuntos**. Este apéndice analiza y define las propiedades, teoremas y principios básicos de la denominada **teoría de conjuntos**.

B.2. Teoría de conjuntos

De forma intuitiva, se define un conjunto como la reunión en un todo de objetos bien definidos y diferenciables entre sí, que se llaman elementos del mismo. Con esa idea intuitiva en la mente, se van a definir de manera formal algunos conceptos básicos de la teoría de conjuntos.

Definición 23 *Si A y B son dos conjuntos, se llama producto cartesiano de A y B, y se escribe AxB al conjunto de los pares ordenados* (a, b)*, donde el primer elemento pertenece al conjunto A y el segundo a B; es decir:*

$$AxB = \{(a, b)/a \in A, b \in B\} \tag{B.1}$$

Definición 24 *Si* $A_1, A_2, \ldots, A_n$ *son conjuntos, se llama producto cartesiano de* $A_1, A_2, \ldots, A_n$ *y se escribe* $A_1 x A_2 x \ldots x A_n$ *al conjunto de las n-uplas ordenadas* $(a_1, a_2, \ldots, a_n)$ *donde* a_i *pertenece a* $A_i, \forall i = 1, \ldots, n$*, es decir:*

$$A_1 x A_2 x \ldots x A_n = \{(a_1, a_2, \ldots, a_n)/a_i \in A_i, \forall i = 1, \ldots, n\} \tag{B.2}$$

Definición 25 *Una relación R entre dos conjuntos A y B (o relación de A en B) es un subconjunto del producto cartesiano AxB.*

Definición 26 *Si $R \subseteq AxB$ es una relación y $(a,b) \in R$, se dice que a está relacionado con b mediante la relación R y se escribe aRb. Es decir:*

$$aRb \Longleftrightarrow (a,b) \in R \tag{B.3}$$

Si $(a,b) \notin R$ se dice que a no está relacionado con b y se escribe $a\neg Rb$.

Definición 27 *Sea R una relación en un conjunto A.*

1. *Se dice que R es reflexiva si, y sólo si, todo elemento de A esta relacionado con el mismo.*
2. *Se dice que R es simétrica si, y sólo si, aRb implica bRa.*
3. *Se dice que R es antisimétrica si, y sólo si, aRb y bRa implica a=b.*
4. *Se dice que R es transitiva si, y sólo si, aRb y bRc implica aRc.*

Definición 28 *Sea R una relación de A en B. Se denomina* **relación inversa** *de R y se denota como R^{-1} a la relación de B en A definida de la siguiente forma:*

$$R^{-1} = \{(b,a) \in BxA / (a,b) \in R\}$$

Definición 29 *Sean R_1 una relación de A en B y R_2 una relación de B en C. Se denomina* **relación compuesta** *de R_1 y R_2 y se denota como R_1R_2 a la relación de A en C definida de la siguiente forma:*

$$R_1R_2 = \{(a,c) \in AxC / \exists b \in B, (a,b) \in AxB \wedge (b,c) \in BxC\}$$

B.3. Relaciones de equivalencia

Definición 30 *Una relación R en un conjunto A es de equivalencia si y sólo si es reflexiva, simétrica y transitiva.*

Definición 31 *Si R es una relación de equivalencia en A y a es un elemento cualquiera de A, se llama clase de a y se escribe [a] al siguiente subconjunto de A:*

$$[a] = \{x \in A / xRa\} \tag{B.4}$$

es decir, el subconjunto formado por todos los elementos de A que están relacionados con a. Cualquier elemento de la clase [a] se llama representante de la clase.

Definición 32 *Si R es una relación de equivalencia en un conjunto A, al conjunto formado por todas las clases de equivalencia se le llama conjunto cociente de A respecto de la relación R y se representa mediante A/R, es decir:*

$$A/R = \{[a] / a \in R\} \tag{B.5}$$

Definición 33 *Una partición en un conjunto A es una familia de subconjuntos no vacíos, disjuntos dos a dos, y tales que la unión de todos ellos es A.*

Teorema 1 *Si R es una relación de equivalencia en un conjunto A, el conjunto cociente A/R es una partición en A.*

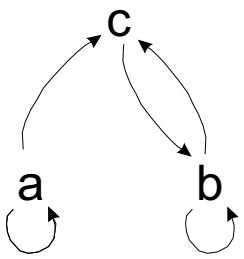


Figura B.1: Digrafo de una relación

B.4. Representación de una relación

Para llevar a cabo la representación de una relación se pueden utilizar dos formalismos, la matriz de la relación y el digrafo de una relación. Ambos formalismos sólo se pueden utilizar para representar relaciones entre conjuntos finitos.

B.4.1. Matriz de una relación

Definición 34 *Si A y B son dos conjuntos finitos, no vacíos, con m y n elementos respectivamente* $A{=}a_1, \ldots, a_m$ *y* $B{=}b_1, \ldots, b_n$ *y R una relación de A en B, llamaremos* **matriz de la relación R** *y la representaremos por* M_R *a la matriz de orden mxn siguiente:*

$$M_R = (m_{ij}) donde\ m_{ij} = \begin{cases} 1 & si\ (a_i, b_j) \in R \\ 0 & si\ (a_i, b_j) \notin R \end{cases} \tag{B.6}$$

Observación 4 *La matriz de una relación caracteriza a la relación y, por tanto, conociendo la relación conocemos la matriz y viceversa.*

Observación 5 *Si la relación R es una relación de equivalencia, su matriz es simétrica (propiedad conmutativa) y todos los elementos de su diagonal son unos (propiedad reflexiva). Por tanto, sólo deben almacenarse la mitad de los elementos, como en el caso de una matriz triangular.*

B.4.2. Digrafo de una relación

Definición 35 *El* **digrafo de una relación** *R en un conjunto A consiste en una representación gráfica que se obtiene mediante el siguiente método:*

1. *Cada elemento de A se representa mediante un punto en el plano.*

2. *Si xRy, entonces se unen los puntos x e y mediante un arco de curva dirigido de x hacia y. El punto x se denomina extremo inicial y el punto y extremo final.*

Ejemplo 5 *Sea* $A = \{a, b, c\}$ *y la relación R:*

$$R = \{(a, a), (a, c), (b, b), (b, c), (c, b)\}$$

La figura B.1 representa el digrafo de la relación R.

B.5. Cierres de una relación

Sea R una relación cualquiera en un conjunto A. Existe al menos una relación reflexiva, simétrica o transitiva mínima que contiene a dicha relación. Este apartado muestra cómo hallar la mínima relación de equivalencia que contiene a R.

Definición 36 *Se denomina* **cierre reflexivo** *de una relación R y se denota como r(R) a la mínima relación reflexiva que contiene a R.*

Definición 37 *Se denomina* **cierre simétrico** *de una relación R y se denota como s(R) a la mínima relación simétrica que contiene a R.*

Definición 38 *Se denomina* **cierre transitivo** *de una relación R y se denota como t(R) a la mínima relación transitiva que contiene a R.*

Proposición 13 *Sea R una relación en A. Se cumple que:*

1. $r(R) = R \iff R$ *es reflexiva*
2. $s(R) = R \iff R$ *es simétrica*
3. $t(R) = R \iff R$ *es transitiva*
4. $r(r(R)) = r(R)$, $s(s(R)) = s(R)$ *y* $t(t(R)) = t(R)$

Teorema 2 *Sea R una relación en A y* $\Delta = \{(a, a) : a \in A\}$. *Se cumple que:*

1. $r(R) = R \cup \Delta$
2. $s(R) = R \cup R^{-1}$
3. $t(R) = \bigcup_{k=1}^{\infty} R^k$

Teorema 3 *Sea R una relación en A y* $|A| = n$, *entonces* $t(R) = \bigcup_{k=1}^{n} R^k$.

Proposición 14 *Sea R una relación en A y* $|A| = n$ *y* M_R *la matriz de la relación R. Sean* $M_{r(R)}$, $M_{s(R)}$ *y* $M_{t(R)}$ *las matrices de los cierres reflexivo, simétrico y transitivo, respectivamente. Se cumple que:*

$$M_{r(R)} = M_R + I$$
$$M_{s(R)} = M_R + M_R^t$$
$$M_{t(R)} = \sum_{k=1}^{n} M_R^k$$

B.6. Relaciones de orden

Definición 39 *Una relación R en un conjunto A es de orden si y sólo si es reflexiva, antisimétrica y transitiva. Para especificar una relación de orden se suele utilizar la notación <=.*

Definición 40 *Un conjunto ordenado es un par (A,R) donde A es un conjunto y R es una relación de orden definida en A.*

Definición 41 *Si R es una relación en un conjunto A, se dice que dos elementos a y b de A son comparables por la relación R si aRb o bRa.*

Definición 42 *Si R es una relación de orden en A en la cuál todo par de elementos de A son comparables por R, se dice que la relación R es un orden total en A. En ese caso, se dice que el conjunto A está totalmente ordenado. Un orden no total se denomina orden parcial.*

Definición 43 *Sea A un conjunto ordenado y $S \subset A$ un subconjunto no vacío de A.*

- *Se dice que un elemento m es* **máximo** *en S si y sólo si $\forall x \in S$ se cumple que $x <= m$.*
- *Se dice que un elemento m es* **mínimo** *en S si y sólo si $\forall x \in S$ se cumple que $m <= x$.*
- *Se dice que un elemento c de A es* **cota superior** *de S si y sólo si $\forall x \in S$ se cumple que $x <= c$.*
- *Se dice que un elemento c de A es* **cota inferior** *de S si y sólo si $\forall x \in S$ se cumple que $c <= x$.*
- *Se dice que un elemento s de A es* **supremo** *de S si y sólo si s es cota superior de S y $\forall c$, cota superior de S se cumple que $s <= c$.*
- *Se dice que un elemento i de A es* **ínfimo** *de S si y sólo si i es cota inferior de S y $\forall c$, cota inferior de S se cumple que $c <= i$.*

Teorema 4 *Sea A un conjunto ordenado y $S \subset A$ un subconjunto de A. El máximo y el mínimo de S, en caso de existir, son únicos.*

Teorema 5 *Sea A un conjunto ordenado y $S \subset A$ un subconjunto de A. El supremo e ínfimo, en caso de existir, son únicos.*

B.7. Representación de una relación de orden. Diagrama de Hasse

En el caso de que se deseen representar relaciones de orden, se puede utilizar el denominado diagrama de Hasse, que es similar al digrafo de la relación, pero con las siguientes simplificaciones:

1. Cada elemento del conjunto se representa mediante un punto en el plano, pero si xRy, entonces x se representa por debajo de y. De este modo, se suprimen las direcciones de los arcos del digrafo, ya que todas son en sentido ascendente.
2. Debido a que las relaciones de orden son reflexivas, se suprimen los arcos que unen un mismo punto.
3. Debido a que las relaciones de orden son transitivas, se eliminan aquellos arcos obtenidos al hallar el cierre transitivo del resto.
4. Se utilizan segmentos para unir los puntos en lugar de arcos de curva.

Ejemplo 6 *Sea $A = \{a, b, c, d\}$ y la relación de orden R:*

$$R = \{(a,a), (a,c), (a,d), (b,b), (b,c), (b,d), (c,c), (c,d), (d,d)\}$$

La figura B.2 representa el diagrama de Hasse de la relación R.

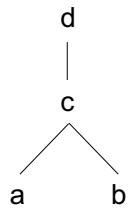


Figura B.2: Diagrama de Hasse

B.8. Retículos

Definición 44 *Un retículo consiste en un conjunto ordenado $(R, <=)$, que cumple que $\forall x, y \in R$:*

1. *existe supremo de x, y*
2. *existe ínfimo de x, y*

Observación 6 *Si dos elementos son comparables, el supremo de ambos es el mayor y el ínfimo el menor.*

Observación 7 *Todo conjunto totalmente ordenado es un retículo, ya que cualquier par de elementos son comparables y, por tanto, el supremo es el mayor de ambos y el ínfimo el menor.*

Definición 45 *Un retículo es una terna $(R, \vee, \wedge)$ donde R es un conjunto que tiene dos operaciones internas binarias denominadas disyunción ($\vee$) y conjunción ($\wedge$), que cumplen las siguientes propiedades:*

1. *idempotente: $x \vee x = x, x \wedge x = x; \forall x \in R$.*
2. *conmutativa: $x \vee y = y \vee x, x \wedge y = y \wedge x; \forall x, y \in R$.*
3. *asociativa: $x \vee (y \vee z) = (x \vee y) \vee z, x \wedge (y \wedge z) = (x \wedge y) \wedge z; \forall x, y, z \in R$.*
4. *absorción: $x \vee (x \wedge y) = x, x \wedge (x \vee y) = x; \forall x, y \in R$.*

Proposición 15 *Las dos definiciones 44 y 45 de retículos son equivalentes.*

Definición 46 *Un retículo es acotado si y sólo si tiene máximo y mínimo. Al máximo se le suele denominar elemento unidad o universal y al mínimo elemento nulo y se pueden representar como 1 y 0, respectivamente.*

B.8.1. Propiedades de los retículos

Proposición 16 *Sea $(R, <=)$ un retículo. $\forall x, y, z \in R$, se cumplen las siguientes propiedades:*

$$x <= z \wedge y <= z \Longleftrightarrow supx, y <= z$$
$$z <= x \wedge z <= y \Longleftrightarrow z <= infx, y$$
$$x <= y \wedge z <= u \Longrightarrow supx, z <= supy, u \wedge infx, z <= infy, u$$
$$x <= y \Longrightarrow supx, z <= supy, z \wedge infx, z <= infy, z$$
$$x <= y \Longleftrightarrow supx, y = y$$
$$x <= y \Longleftrightarrow infx, y = x$$
$$supx, y = y \Longleftrightarrow infx, y = x$$

Proposición 17 *Si $(R, <=)$ es un retículo acotado, se cumplen las siguientes propiedades:*

1. $\forall a \in R, 0 <= a <= 1$
2. $\forall a \in R, supa, 0 = a \wedge inf a, 0 = 0$
3. $\forall a \in R, supa, 1 = 1 \wedge inf a, 1 = a$

Proposición 18 *Todo retículo finito R es acotado.*

B.8.2. Subretículos

Definición 47 *Sea $(R, <=)$ un retículo y $A \subset R$ un subconjunto de R. Se dice que $(A, <=)$ es un subretículo de $(R, <=)$ si y sólo si se cumplen las siguientes condiciones:*

1. *$(A, <=)$ es un retículo.*
2. *$\forall x, y \in A, sup_A x, y = sup_R x, y$ e $inf_A x, y = inf_R x, y$.*

Bibliografía

[ABS02] L. Amar, A. Barak, and A. Shiloh. The MOSIX parallel I/O system for scalable I/O performance. In *P*roceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002), Cambridge, MA, pages 495–500, November 2002.

[AC87] P. Agre and D. Chapman. PENGI: An implementation of a theory of activity. In *P*roceedings of the 6th National Conference on Artificial Intelligence (AAAI-87), pages 268–272, 1987.

[ACC+00] R. Aiken, M. Carey, B. Carpenter, I. Foster, C. Lynch, J. Mambreti, R. Moore, J. Strasnner, and B. Teitelbaum. Network policy and services: A report of a workshop on middleware. Technical Report RFC 2768, IETF, 2000.

[ACG86] S. Ahuja, N. Carreiro, and D. Gelernter. Linda and friends. *I*EEE Computer, 19(8):26–34, August 1986.

[ACP95] T. Anderson, D. Culler, and D. Patterson. A case for NOW (networks of workstations). *IEEE Micro*, pages 54–64, Feb 1995.

[ADN+96] T. Anderson, M. Dahlin, J. Neefe, D. Patterson, D. Roselli, and R. Wang. Serverless Network File Systems. *ACM* Transactions on Computer Systems, 14(1):41–79, Feb 1996.

[ADS99] E. M. Atkins, E. H. Durfee, and K. G. Shin. Autonomous flight with CIRCA-II. In *A*utonomous Agents'99 Workshop on Autonomy Control Software, May 1999.

[AFC00] C. P. Azevedo, B. Feiju, and M. Costa. Control centres evolve with agent technology. *IEEE* Computer Applications in Power, 13(3):48–53, 2000.

[AG92] N. M. Avouris and L. Gasser. *D*istributed Artificial Intelligence: Theory and Praxis. Volume 5 of Computer and Information Science. Kluwer Academic Publisher, Boston, MA, 1992.

[AHT90] J. F. Allen, J. Hendler, and A. Tate, editors. *R*eadings in Planning. Morgan Kaufmann Publishers: San Mateo, CA, 1990.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. Mining association rules between sets of items in large databases. In *T*he 1993 ACM SIGMOD International Conference on Management of Data, 1993.

[Amd67] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *A*FIPS 1967 Spring Joint Computer Conference, volume 30, pages 483–485, April 1967.

[And00] G. R. Andrews. *F*oundations of Multithreaded Parallel and Distributed Programming. Addison-Wesley, 2000.

[ANS98] American National Standard. Knowledge Interchange Format. *Draft Proposed American National Standard (dpANS), NCITS.T2/98-004* , 1998.

[Ari90] Ben Ari. *P*rinciples of Concurrent and Distributed Programming. Prentice-Hall, 1990.

[ARP93] ARPA knowledge sharing initiative. specification of the KQML agent-communication language. *External Interfaces Working Grop working paper*, 1993.

[AS96] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE* Transactions on Kkowledge and Data Engg., 8:962–969, 1996.

[AS99] J. Almond and D. Snelling. UNICORE: uniform access to supercomputing as an element of electronic commerce. *F*GCS, 15(5-6):539–548, October 1999.

[Bac86] M. J. Bach. *T*he Design of the UNIX Operating System. Prentice Hall Inc., Englewood Cliffs, NJ, 1986.

[Bau99] Joachim Baumann. *C*ontrol algorithms for Mobile Agents. PhD thesis, Universidad de Stuttgart, 1999.

[BB99] Mark Baker and Rajkumar Buyya. *C*luster Computing at a Glance, chapter Requirements and General Issues. Prentice Hall, 1999. Chapter belonging to [Buy99a].

[BBS+94] Robert Bennett, Kelvin Bryant, Alan Sussman, Raja Das, and Joel Saltz. Jovian: A framework for optimizing parallel I/O. In *P*roceedings of the Scalable Parallel Libraries Conference, pages 10–20. IEEE Computer Society Press, October 1994.

[BCF+98] S. Brunett, Karl Czajkowski, Steven Fitzgerald, Ian T. Foster, Andrew E. Johnson, Carl Kesselman, Jason Leigh, and Steven Tuecke. Application experiences with the Globus Toolkit. In *P*roc. 7th IEEE Symp. on High Performance Distributed Computing, pages 81–89, 1998.

[BDK95] A. Baratloo, P. Dasgupta, and Z. Kedem. Calypso: A novel software system for fault-tolerant parallel processing on distributed platforms. In *P*roceedings of IEEE International Symposium on High-Performance Distributed Computing, 1995.

[BDKJT97] F. M. T. Brazier, B. M. Dunin-Keplicz, N. R. Jennings, and J. Treur. DESIRE: Modelling multi-agent systems in a compositional formal framework. *I*nt Journal of Cooperative Information Systems, 6(1):67–94, 1997.

[Bel92] Richard K. Belew. Un ruolo per la scienza dei calcolatori nella vita artificiale. *S*istemi Intelligenti, 4(2):1–9, August 1992.

[BEO] The Beowulf Project. *http://www.beowulf.org.*

[BFK+00] M. D. Beynon, R. Ferreira, T. Kurc, A. Sussman, and J. Saltz. DataCutter: Middleware for filtering very large scientific datasets of archival storage systems. In *P*roceedings of the 2000 Mass Storage Systems Conference, pages 119–133, College Park, MD, Mar 2000. IEEE Computer Society Press.

[BFY96] M.A. Baker, G.C. Fox, and H.W. Yau. Review of Cluster Management Software. *NHSE Review*, May 1996.

[BG88a] D. Bitton and J. Gray. Disk shadowing. In *P*roceedings of the 14th International Conference on Very Large Databases, pages 331–338, August 1988.

[BG88b] A. H. Bond and L. Gasser. *A*n analysis of problems and research in DAI, pages 3–36. Springer-Verlag, 1988. Article belonging to [BG88c].

[BG88c] A. H. Bond and L. Gasser, editors. *R*eadings in Distributed Artificial Intelligence. Morgan Kauffmann, 1988.

[BG92] A. H. Bond and L. Gasser. A subject-indexed bibliography of distributed artificial intelligence. *I*EEE Transaction on Systems, Man, and Cybernetics, 22(6):1260–1281, November 1992.

[BGST93] Michael L. Best, Adam Greenberg, Craig Stanfill, and Lewis W. Tucker. CMMD I/O: A parallel UNIX I/O. In *P*roceedings of the Seventh International Parallel Processing Symposium, pages 489–495, 1993.

[BHK+91] Mary Baker, John H. Hartman, Michael D. Kupfer, Ken W. Shirriff, and John Ousterhout. Measurements of a distributed file system. In *P*roceedings of the Thirteenth Symposium on Operating Systems Principles, pages 198–211, October 1991.

[BIP88] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 4:349–355, 1988.

[BL92] Ralph Buttler and Ewing Lusk. User's guide to the p4 parallel programmimg system. Technical Report ANL92 /17, Argonne National Laboratory, October 1992.

[BMB+00] Jean Bacon, Ken Moody, John Bates, Richard Hayton, Chaoying Ma, Andrew McNeil, Oliver Seidel, and Mark Spiteri. Generic support for distributed applications. *C*omputer, 33(3):68–76, 2000.

[BMRW98] C. Baru, R. Moore, A. Rajasekar, and M. Wan. The SDSC storage resource broker. In *P*roceedings of CASCON'98, Toronto, Canada, 1998.

[BMS96] Peter Brezany, Thomas A. Mueck, and Erich Schikuta. A software architecture for massively parallel Input-Output. In *T*hird International Workshop PARA'96 (Applied Parallel Computing - Industrial Computation and Optimization), volume 1186, pages 85–96, Lyngby, Denmark, 1996. Springer-Verlag.

[BN84] A. D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[BO98] A. Barak and O. La'adan O. The MOSIX multicomputer operating system for high performance cluster computing. *J*ournal of Future Generation Computer Systems, 13(4-5):361–372, March 1998.

[Boo91] Grady Booch. *O*bject-Oriented design with applications. Benjamin-Cummings, 1991.

[Bra87] *I*ntentions, Plans and Practical Reason. Harvard University Press: Cambridge, MA, 1987.

[Bra90] M. E. Bratman. *W*hat is intention?, pages 15–32. 1990. Article belonging to [CMP90].

[Bro86] R. A. Brooks. A robust layered control system for a mobile root. *I*EEE Journal of Robotics and Automation, 2(1):14–23, 1986.

[Bro90] R. A. Brooks. Elephants don't play chess. *D*esigning Autonomous Agents, pages 3–15, 1990.

[Bro91a] R. A. Brooks. Intelligence without reason. *A*rtificial Intelligence, 47:139–159, 1991.

[Bro91b] R. A. Brooks. Intelligence without representation. *A*rtificial Intelligence, 47:139–159, 1991.

[Bru91] Jose C. Brustoloni. Autonomous agents: Characterization and requirements. *Technical Report CMU-CS-91-204, School of Computer Science, Carnegie Mellon University*, November 1991.

[BS92] B. Burmeister and K. Sundermeyer. Cooperative problem solving guided by intentions and perception. In E. Werner and Y. Demazeau, editors, *D*ecentralized AI 3 - Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91), pages 77–92. Elsevier Science Publisher B.V: Amsterdam, The Netherlands, 1992.

[Buy99a] Rajkumar Buyya, editor. *H*igh Performance Cluster Computing: Architectures and Systems. Volume 1. Prentice Hall, 1999.

[Buy99b] Rajkumar Buyya, editor. *H*igh Performance Cluster Computing: Programming and Applications. Volume 2. Prentice Hall, 1999.

[CA86] D. Chapman and P. Agre. *A*bstract reasoning as emergent from concrete activity, pages 411–424. 1986. Article belonging to [GL86].

[CACR95] P. E. Crandall, R. A. Aydt, A. A. Chien, and D. A. Reed. Characterization of a suite of Input/Output intensive applications. In *P*roceedings of Supercomputing'95, 1995.

[CBH+94] A. Choudhary, R. Bordawekar, M. Harry, R. Krishnaiyer, R. Ponnusamy, T. Singh, and R. Thakur. PASSION: Parallel and scalable software for Input-Output. Technical Report SCCS–636, NPAC, Syracuse University, September 1994.

[CC94] P. R. Cohen and A. Cheyer. *A*n Open Agent Architecture, pages 1–8. 1994. Article belonging to [Etz94].

[CF94] P. Corbett and D. Feitelson. Design and implementation of the Vesta parallel file system. In *P*roceedings of the Scalable High-Performance Computing Conference, pages 63–70, 1994.

[CF96] Peter F. Corbett and Dror G. Feitelson. The Vesta parallel file system. *A*CM Transactions on Computer Systems, 14(3):225–264, August 1996.

[CFE+93] M. Cutkosky, L. Fikes, R. Engelmore, M. Genesereth, M. Mark, W. Gruber, T. Tenenbaum, and J. Weber. PACT: An experiment in integrating concurrent engineering systems. *I*EEE Transactions on Computer, 26(1):28–37, 1993.

[CFF+92] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Wiederhold. An overview of KQML: A knowledge query and manipulation language. Technical report, Computer Science Department. Stanford University, April 1992.

[CFF+95] P. Corbett, D. Feitelson, S. Fineberg, Y. Hsu, B. Nitzberg, J. Prost, M. Snir, B. Traversat, and P. Wong. Overview of the MPI-IO parallel I/O interface. In *P*roceedings of the Third Workshop on I/O in Parallel and Distributed Systems, IPPS '95, Santa Barbara, CA, pages 1–15, April 1995.

[CFKL95] P. Cao, E. W. Felten, A. R. Karlin, and K. Li. A study of integrated prefetching and caching strategies. In *P*roceedings of the Joint Intl. Conf. on Measurement and Modeling of Computer Systems, pages 188–197. ACM Press, May 1995.

[CFL94] Pei Cao, Edward W. Felten, and Kai Li. Implementation and performance of application-controlled file caching. In *O*perating Systems Design and Implementation, pages 165–177, 1994.

[CFP+95] Peter F. Corbett, Dror G. Feitelson, Jean-Pierre Prost, George S. Almasi, Sandra Johnson Baylor, Anthony S. Bolmarcich, Yarsun Hsu, Julian Satran, Marc Snir, Robert Colao, Brian Herr, Joseph Kavaky, Thomas R. Morgan, and Anthony Zlotek. Parallel file systems for the IBM SP computer. *IBM Systems Journal*, 34(2):222–248, 1995.

[CFPS93] P. Corbett, D. Feitelson, J. Prost, and Johnson S. Parallel access to files in the Vesta file system. In *P*roc. of the 15th. Int. Symp. on Operating Systems Principles, pages 472–481. ACM, 1993.

[CGKP90] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson. An evaluation of redundant arrays of disks using an Amdahl 5890. In *P*roceedings of SIGMETRICS, pages 74–85, May 1990.

[CHKM93] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *I*EEE Proceedings of the 20th International Symposium on Computer Architecture, San Diego, California, pages 2–13, 1993.

[CHLY95] Ming-Syan Chen, Hui-I Hsiao, Chung-Sheng Li, and Philip S. Yu. Using rotational mirrored declustering for replica placement in a disk-array-based video server. In *P*roceedings of the Third ACM Conference on Multimedia, pages 121–130, November 1995.

[CHN+96] D. Cheung, J. Han, V. Ng, A. Fu, and Y. Fu. A fast distributed algorithm for mining association rules. In *C*onf. of Parallel and Distributed Information Systems, 1996.

[Cho99] Yong E. Cho. *E*fficient resource utilization for parallel I/O in cluster environments. PhD thesis, University of Illinois at Urbana-Champaign, 1999.

[CHR] Christian Borgelt's homepage. *http://fuzzy.cs.uni-magdeburg.de/ borgelt.*

[CIRT00] P. H. Carns, W. B. Ligon III, R. B. Ross, and R. Thakur. PVFS: A parallel file system for linux clusters. In *P*roceedings of the 4th Annual Linux Showcase and Conference, pages 317–327, October 2000.

[CJF93] P. Corbett, S. Johnson, and D. Feitelson. Overview of the Vesta parallel file system. *A*CM Computer Architecture News, 21(5):7–15, Dec 1993.

[CK89] G. Copeland and T. Keller. A comparison of high-availability media recovery techniques. In *P*roceedings of the ACM Conference on Management of Data (SIGMOD), pages 98–109, 1989.

[CK93] Thomas H. Cormen and David Kotz. Integrating theory and practice in parallel file systems. In *P*roceedings of the 1993 DAGS/PC Symposium, Hanover, NH, pages 64–74, June 1993.

[CKL94] Y. Cengeloglu, S. Khajenoori, and D. Linton. DYNACLIPS (DYNAmic CLIPS): A dynamic knowledge exchange tool for intelligent agents. In *P*roceedings of the Third CLIPS Conference at NASA's Johnson Space Center, September 1994.

[CKV93] Kenneth M. Curewitz, P. Krishnan, and Jeffrey Scott Vitter. Practical prefetching via data compression. In *P*roceedings of the ACM-SIGMOD Conference on Management of Data, pages 257–266, May 1993.

[CL90a] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[CL90b] P. R. Cohen and H. J. Levesque. *R*ational interaction as the basis for communication, pages 221–256. 1990. Article belonging to [CMP90].

[CL92] N. Carver and V. Lesser. The evolution of blackboard control architectures. Technical report, University of Massachusetts, Amherst, October 1992.

[CLG+94] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2), June 1994.

[CM01] Adam Cheyer and David Martin. The Open Architecture. *A*utonomous Agents and Multi-Agent Systems, 4:143–148, 2001.

[CMP90] P. R. Cohen, J. Morgan, and M. E. Pollack, editors. *I*ntentions in Communication. The MIT Press: Cambridge, MA, 1990.

[CMS83] S. Cammarata, D. McArthur, and I. Steeb. Strategies of cooperation in distributed problem solving. In *P*roceedings of the Eighth International Joint Conference on Artificial Intelligence (IJCAI-83). Menlo Park, Calif., pages 767–770, 1983.

[CN96] D. Cockburn and J. Nick. *A*RCHON: A distributed artificial intelligence system for industrial applications, pages 319–344. 1996. Article belonging to [OJ96].

[Coe94] Michael H. Coen. SodaBot: A software agent environment and construction system. Technical Report AITR-1493, 1994.

[Con63] Conway63. A multiprocessor system design. In *P*roceedings AFIPS Fall Joint Computer Conf., volume 24, pages 139–146. Spartan Books, New York, 1963.

[Cor91] D. Corkill. Blackboard systems. *A*I Expert, 6(9):40–47, September 1991.

[CPdM+96] J. Carretero, F. Pérez, P. de Miguel, F. García, and L. Alonso. ParFiSys: A parallel file system for MPP. *ACM Operating Systems Review*, 30(2):74–80, April 1996.

[CPdM+97a] J. Carretero, F. Pérez, P. de Miguel, F. García, and L. Alonso. Evaluating ParFiSys: a high-performance and distributed file system. *J*ournal of Systems Architecture, 43:533–542, 1997.

[CPdM+97b] J. Carretero, F. Pérez, P. de Miguel, F. García, and L. Alonso. Improving the performance of parallel file systems. *P*arallel Computing, 23:525–542, 1997.

[CR93] C.M. Chen and N. Roussopoulos. Adaptive database buffer allocation using query feedback. In *P*roceedings of the 19th. International Conference on Very Large Data Bases, pages 342–353, 1993.

[Cro89] Thomas W. Crockett. File concepts for parallel I/O. In *P*roceedings of Supercomputing '89, pages 574–579, 1989.

[CTB+96] A. Choudhary, R. Thakur, R. Bordawekar, S. More, and S. Kutipidi. PASSION optimized I/O for parallel applications. *I*EEE Computer, 29(6):70–78, June 1996.

[CVRS94] F. Cheng, P. Vaughan, D. Reese, and A. Skjellum. *T*he Unify System. Engineering Research Center, Mississippi State University, 1 1994.

[CY89] D. W. Cornell and P. S. Yu. Integration of buffer management and query optimization in relational database environment. In *P*roceedings of the 15th VLDB Conference, pages 247–255, Amsterdam, August 1989.

[DB96] T. Dan and W. Birmingham. An attribute-space representation and algorithm for concurrent engineering. *A*rtificial Intelligence for Engineering Analysis and Manufacturing, 10(1):21–35, 1996.

[DBM+92] L. Dent, J. Boticario, J. McDermott, T. Mitchell, and D. Zabowski. A personal learning apprentice. In *P*roceedings of the Tenth National Conference on Artificial Intelligence, pages 96–103, 1992.

[DCF95] A. Drogoul, B. Corbara, and D. Fresneau. MANTA : New experimental results on the emergence of (artificial) ant societies. *A*rtificial Societies: the computer simulation of social life, 1995.

[DCP95] Feitelson D., P. Corbett, and J. Prost. Performance of the Vesta parallel file system. In *P*roceedings of the Ninth International Parallel Processing Symposium, pages 150–158, April 1995.

[DdR91] E. DeBenedictis and J.M. del Rosario. Scalable I/O. Technical Report nCUBE-TR001-911015, nCUBE Corporation, October 1991.

[DdR92] E. DeBenedictis and J. M. del Rosario. nCUBE parallel I/O software. In *E*leventh Annual IEEE International Phoenix Conference on Computers and Communications (IPCCC), pages 117–124, Apr 1992.

[DE94] W. H. E. Davies and P. Edwards. Agent-K: An integration of AOP and KQML. In *P*roceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), November 1994.

[Dei87] H. M. Deitel. *I*ntroducción a los Sistemas Operativos. Addison-Wesley Iberoamericana. México, 1987.

[Den78] D. C. Dennett. *B*rainstorms. MIT Press, Cambridge, MA, 1978.

[Den87] D. C. Dennett. *T*he Intentional Stance. MIT Press, Cambridge, MA, 1987.

[DF92] A. Drogoul and J. Ferber. Multi-agent simulation as a tool for modeling societies: Applications to social differentiation in ant colonies. In C. Castelfranchi and E. Werner, editors, *A*rtificial Social Systems - Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-92)(LNAI Volume 830), pages 3–23. Springer-Verlag: Heidelberg, Germany, 1992.

[DGMP95] J.J. Dongarra, G. A. Geist, R.J. Manchek, and P.M. Papadopoulos. Adding context and static groups into PVM. *h*ttp://www.epm.ornl.gov/pvm/context.ps, July 1995.

[DL88] E. H. Durfee and V. R. Lesser. *U*sing partial global plans to coordinate distributed problem solvers, pages 285–293. Springer-Verlag, 1988. Article belonging to [BG88c].

[DL89] E. H. Durfee and V. Lesser. *N*egotiating task decomposition and allocation using partial global planning, pages 229–244. 1989. Article belonging to [GH89].

[DLC87] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Coherent cooperation among communicating problem solvers. *I*EEE Transactions on Computers, 36:1275–1291, 1987.

[DLC89] E. H. Durfee, V. R. Lesser, and D. D. Corkill. Trends in cooperative distributed problem solving. *I*EEE Transactions on Knowledge and Data Engineering, 1(1), March 1989.

[dR92a] Juan Miguel del Rosario. Disk striping with parallel I/O. Technical Report nCUBE-TR002-920104, nCUBE Corporation, January 1992.

[dR92b] Juan Miguel del Rosario. High performance parallel I/O on the nCUBE 2. *I*nstitute of Electronics, Information and Communication Engineers (IEICE) Transactions, Japan, aug 1992.

[DR94] E. Durfee and J. Rosenschein. Distributed problem solving and multi-agent systems: Comparisons and examples. In *P*roceedings of the Thirteenth International Distributed Artificial Intelligent Workshop, 1994.

[dRBC93] J. del Rosario, R. Bordawekar, and A. Choundary. Improved parallel I/O via a two-phase run-time access strategy. *A*CM Computer Architecture News, 21(5):31–38, Dec 1993.

[dRC94] J.M. del Rosario and A.N. Choudhary. High-performance I/O for massively parallel computers: Problems and prospects. *I*EEE Computer, 27(3):59–68, 1994.

[DSE88] P. Dibble, M. Scott, and C. Ellis. BRIDGE: A high performance file system for parallel processors. In *P*roceedings of the IEEE Eighth ICDCS, pages 154–161. IEEE, Jun 1988.

[DSW97] K. Decker, K. Sycara, and M. Williamson. Middle agents for the internet. In *P*roceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97), Menlo Park. Calif., pages 578–583, 1997.

[Dur87] E. H. Durfee. *A* Unified Approach to Dynamic Coordination: Planning Actions and Interactions in a Distributed Problem Solving Network. PhD thesis, Department of Computer and Information Science, University of Massachusetts, 1987.

[Dur96] E. H. Durfee. *P*lanning in Distributed Artificial Intelligence, pages 231–245. 1996. Article belonging to [OJ96].

[Eck90] H. Eckardt. Investigation of distributed disk I/O concepts. Technical Report ESPRIT PUMA, Siemens, August 1990.

[EH86] E. A. Emerson and J. Y. Halpern. 'sometimes´and 'not never´revisited: on branching time versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EK89] C. S. Ellis and D. Kotz. Prefetching in file systems for MIMD multiprocessors. In *P*roceedings of the 1989 International Conference on Parallel Processing, pages 306–314, August 1989.

[EK93] R. Esser and R. Knecht. Intel Paragon XP/S— architecture and software environment. In *P*roceedings of Supercomputer '93, Lecture Notes in Computer Science, Mannheim, Germany, 1993.

[EKHM93] C. Elford, C. Kuszmaul, J. Huber, and T. Madhyastha. Scenarios for the Portable Parallel File System, 1993.

[Etz94] O. Etzioni, editor. *S*oftware Agents. AAAI Press, March 1994.

[Etz96] O. Etzioni. The World Wide Web: Quagmire or gold mine? *C*ommunications of the ACM, 39(11):65–68, 1996.

[FA93] G. Ferguson and J. F. Allen. Cooperative plan reasoning for dialogue systems. Technical report, University of Rochester, 1993.

[FCO90] J. Feo, D. Cann, and R. Oldehoeft. A report on the SISAL language project. *J*ournal of Parallel and Distributed Computing, 10:349–366, 1990.

[FD92] J. Ferber and A. Drogoul. *U*sing reactive multi-agents systems in simulation and problem solving, pages 53–80. 1992. Article belonging to [AG92].

[FE89] Richard A. Floyd and Carla S. Ellis. Directory reference patterns in hierarchical file systems. *I*EEE Transactions on Knowledge and Data Engineering, 1(2):238–247, June 1989.

[Fer92a] I. A. Ferguson. *TouringMachines*: An Architecture for Dynamic, Rational, Mobile Agents. PhD thesis, University of Cambridge, UK, 1992.

[Fer92b] I. A. Ferguson. Towards an architecture for adaptative, rational, mobile agents. In E. Werner and Y. Demazeau, editors, *D*ecentralized AI 3 - Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91), pages 249–262. Elsevier Science Publisher B.V: Amsterdam, The Netherlands, 1992.

[Fer95a] I. A. Ferguson. *I*ntegrated control and coordinated behaviour: A case for agent models, pages 203–218. Jan 1995. Article belonging to [WJ95b].

[Fer95b] I. A. Ferguson. On the role of BDI modeling for integrated control and coordinated behavior in autonomous agents. *Applied Artificial Intelligence*, 9(4):421–448, November 1995.

[Fer96] J. Ferber. *R*eactive Distributed Artificial Intelligence: Principles and Applications, pages 287–314. 1996. Article belonging to [OJ96].

[FG96] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *P*roceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag, 1996.

[FGC00] Javier Fernández, Félix García, and Jesús Carretero. Enhancing parallel multimedia servers through new hierarchical disk scheduling algorithms. *V*ector and Parallel Processing - VECPAR 2000. LNCS 1981, pages 89–99, 2000.

[Fis93] K. Fischer. The rule-based Multi-Agent System MAGSY. In *P*roceedings of the 1992 Workshop on Cooperating Knowledge Based Systems CKBS'92 Workshop, 1993.

[Fis94] M. Fisher. A survey of concurrent METATEM – the language and its applications. In D. M. Gabbay and H. J. Ohlbach, editors, *T*emporal Logic - Proceedings of the First International Conference (LNAI Volume 827), pages 480–505. Springer-Verlag: Heidelberg, Germany, 1994.

[FK97] Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *T*he International Journal of Supercomputer Applications and High Performance Computing, 11(2):115–128, Summer 1997.

[FK99] I. Foster and C. Kesselman, editors. *T*he Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1999.

[FL77] R. D. Fennell and V. R. Lesser. Parallelism in artificial intelligence problem solving: A case study of Hearsay II. *I*EEE Trans. on Computers, 26(2):106–119, February 1977.

[FL99] Lori A. Freitag and Raymond M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *P*roceedings of SC99: High Performance Networking and Computing, Portland, OR, November 1999. ACM Press and IEEE Computer Society Press.

[FLM97] Tim Finin, Yannis Labrou, and James Mayfield. KQML as an agent communication language. "Software Agents", MIT Press. Cambridge, 1997.

[Flo86] Richard A. Floyd. Short-term file reference patterns in a UNIX environment. Technical Report Technical Report TR-177, Dept. of Computer Science, University of Rochester, 1986.

[FM99] Roberto A. Flores-Mendez. Towards a standardization of multi-agent system frameworks. *A*CM Crossroads, Special Issue on Intelligent Agents, Association for Computer Machinery, 5(4):18–24, 1999.

[FMP96] K. Fisher, J. Müller, and M. Pischel. *A*GenDA: A general testbed for distributed artificial intelligence applications, pages 401–427. 1996. Article belonging to [OJ96].

[FO71] R. Feiertag and E. Organisk. The Multics Input/Output System. In *P*roceedings of the 3rd Symposium on Operating System Principles, pages 35–41, 1971.

[Fon93] L.Ñ. Foner. What's an agent, anyway? a sociological case stydy. Technical Report Agents Memo 93-01, MIT Media Lab, 1993.

[For81] C. Forgy. OPS5 user's manual. Technical Report CMU-CS-81-135, Carnegie-Mellon University, 1981.

[Fos91] Parallel programming with PCN. Technical Report ANL-91/32, Argonne National Laboratory, University of Chicago, 1991.

[Fos95] I. Foster. *D*esigning and Building Parallel Programs. Addison-Wesley, Reading, Massachusetts, 1995.

[Fos99] Ian Foster. *C*omputational Grids. 1999. Chapter belonging to [FK99].

[Fos01] Ian Foster. The anatomy of the Grid: Enabling scalable virtual organizations. *L*ecture Notes in Computer Science, 2150, 2001.

[FR96] Michael J. Flynn and Kevin W. Rudd. Parallel architectures. *A*CM Computing Surveys, 28(1):67–70, Mar 1996.

[FW99] Roberto A. Flores and Niek J.E. Wijngaards. Primitive interaction protocol for agents in a dynamic environment. In *P*roceedings of the 12th Workshop on Knowledge Acquisition, Modeling and Management (KAW'99), October 1999.

[GA94] Jim Griffioen and Randy Appleton. Reducing file system latency using a predictive approach. In *USENIX* Summer, pages 197–207, 1994.

[Gar98] Félix García. Soluciones al problema de la E/S en sistemas distribuidos y paralelos. Technical Report FIM/105.1/DATSI/98, Universidad Politécnica de Madrid, 1998.

[Gas92] L. Gasser. *D*AI Approaches to coordination, pages 31–51. 1992. Article belonging to [AG92].

[GBD+94a] Al Geist, Adan Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 users guide and reference manual. Technical report, Oak Ridge National Laboratory, May 1994.

[GBD+94b] Al Geist, Adan Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM*: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing. MIT Press, 1994.

[GBH88] L. Gasser, C. Braganza, and N. Herman. *I*mplementing Distributed AI Systems Using MACE, pages 445–450. Kaufmann, San Mateo, CA, 1988. Article belonging to [BG88c].

[GCA] Grand Challenging Applications. *http://www.mcs.anl.gov/projects/grand-challenges.*

[GCC+02] F. García, A. Calderón, J. Carretero, J. Fernández, and J. M. Pérez. Expandable parallel file system using NFS servers. In *P*roceedings of the 5th International Meeting on High Performance Computing for Computational Science (VECPAR 2002), June 2002.

[GCPdM99] F. García, J. Carretero, F. Pérez, and P. de Miguel. High performance cache management for parallel file systems. *L*ectures Notes in Computer Science, 1573:466–479, 1999.

[GCPS02] F. García, A. Calderón, M.S. Pérez, and L.M. Sánchez. Evaluating Expand: A parallel file system using NFS servers. In *P*roceedings of the 6th World Multiconference on Systemics, Cybernetics and Informatics, July 2002.

[GCS96] M. Garijo, A. Cáncer, and J. J. Sánchez. A multiagent system for cooperative network-fault management. In *P*roceedings of the First International Conference on the Practical Applications of Intelligent Agents and Multi-Agent Technology, PAAM-96, pages 279–294, April 1996.

[Geo89] M. Georgeff. *C*ommunication and interaction in multi-agent planning, pages 200–209. 1989. Article belonging to [GH89].

[GF01] Olivier Gutknecht and Jacques Ferber. The MadKit agent platform architecture. *I*nfrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, pages 48–55, March 2001.

[GGL93] N. Galbreath, W. Gropp, and D. Levine. Applications-driven parallel I/O. In *P*roceedings of Supercomputing'93, pages 462–471, 1993.

[GGR88] M. R. Genesereth, M. L. Ginsberg, and J. S. Rosenschein. *C*ooperation without communication, pages 220–226. Springer-Verlag, 1988. Article belonging to [BG88c].

[GH89] L. Gasser and M. Huhns, editors. *D*istributed Artificial Intelligence Volume II. Pitman Publishing: London and Morgan Kaufmann: San Mateo, CA, 1989.

[GHPW90] G.A. Geist, M.T. Heath, B.W. Peyton, and P.H. Worley. PICL: A portable instrumented communication library. Technical Report ORNL/TM-11130, Oak Ridge National Laboratory, July 1990.

[GHW90] Jim Gray, Bob Horst, and Mark Walker. Parity striping of disk arrays: Low-cost reliable storage with acceptable throughput. In *P*roceedings of the 16th International Conference on Very Large Data Bases, pages 148–160, 1990.

[GI89a] M. P. Georgeff and F. F. Ingrand. Decision-making in an embedded reasoning system. In *P*roceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89). Detroit, MI, pages 972–978, 1989.

[GI89b] M. P. Georgeff and F. F. Ingrand. Monitoring and control of spacecraft systems using procedural reasoning. Technical Report 03, Australian Artificial Intelligence Institute, Melbourne, Australia, November 1989.

[Gia88] J. Giarrantano. CLIP's user's guide. *A*rtificial Intelligence Section, Johnson Space Center, NASA, June 1988.

[Gib95] G. Gibson. The Scotch Paralell Storage Systems. Technical Report CMU-CS-95-107, Scholl of Computer Science, Carnegie Mellon University, Pittsburbh, Pennsylvania, 1995.

[Gil58] S. Gill. Parallel programming. *T*he Computer Journal, 1:2–10, Apr 1958.

[GK94] M. R. Genesereth and S. P. Ketchpel. Software agents. *Communications of the ACM*, 37(7):48–53, 1994.

[GL86] M. P. Georgeff and A. L. Lansky. *R*easoning About Actions And Plans - Proceedings of the 1986 Workshop. Morgan Kaufmann Publishers: San Mateo, CA, 1986.

[GL87] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *P*roceedings of the 6th National Conference on Artificial Intelligence (AAAI-87), pages 677–682, 1987.

[GL97] William Gropp and Ewing Lusk. Why are PVM and MPI so different? Technical Report Technical Report PREPRINT ANL/MCS-P667-0697, Mathematics and Computer Science Division, Argonne National Laboratory, June 1997.

[GLO] The Globus Project. *http://www.globus.org*.

[GMM+01] John R. Graham, Daniel McHugh, Michael Mersic, Foster McGeary, M. Victoria Windley, David Cleaver, and Keith S. Decker. Tools for developing and monitoring agents in distributed multi-agent systems. *I*nfrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, pages 12–27, March 2001.

[GN87] M. R. Genesereth and N. Nilsson. *L*ogical Foundations of Artificial Intelligence. Morgan Kaufmann Publishers: San Mateo, CA, 1987.

[GNA+97] Garth A. Gibson, David Nagle, Khalil Amiri, Fay W. Chang, Eugene M. Feinberg, Howard Gobioff, Chen Lee, Berend Ozceri, Erik Riedel, David Rochberg, and Jim Zelenka. File server scaling with network-attached secure disks. In *P*roceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (Sigmetrics '97). ACM Press, Jun 1997.

[GP93] Garth A. Gibson and David A. Patterson. Designing disk arrays for high data reliability. *Journal of Parallel and Distributed Computing*, 1993.

[Gra95] R. S. Gray. Agent Tcl: A transportable agent system. In *F*ourth International Conference on Information and Knowledge Management (CIKM 95), Baltimore, Maryland, December 1995.

[Gra96] R. S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In M. Diekhans and M. Roseman, editors, *F*ourth Annual Tcl/Tk Workshop (TCL 96), pages 9–23, Monterey, CA, 1996.

[Gru93] T. R. Gruber. A translation approach to portable ontology specification. *K*nowledge Acquisition, 5(2):199–220, 1993.

[GS86] B. J. Grosz and C. L. Sidner. Attention, intentions and the structure of discourse. *C*omputational Linguistics, 12(3):175–204, 1986.

[GS94] Michael Gray and D. Patrick Studdard. Incorporation emotional agents into the cooperation mechanism of the Accord System. In *P*roceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), November 1994.

[GS96] L. Garrido and K. Sycara. Multiagent meeting scheduling: Preliminary experimental results. In *P*roceedings of the Second International Conference on Multiagent Systems, pages 95–102, 1996.

[GWF+97] A.S. Grimshaw, W.A. Wulf, J.C. French, A.C. Weaver, and P.F. Reynolds. The Legion vision of a worldwide virtual computer. *C*ommunications of the ACM, 40(1), January 1997.

[Had93] A. Haddadi. A hybrid architecture for multi-agent systems. In S. M. Deen, editor, *P*roceedings of the 1993 Workshop on Cooperating Knowledge Based Systems (CKBS-93), pages 13–26, 1993.

[Hal87] Paul R. Halmos. *N*aive Set Theory. Springer Verlag, 1987.

[Hau94] H. Haugeneder. IMAGINE final project report. *IMAGINE Technical Report Series*, 1994.

[HC95] B. Huberman and S. H. Clearwater. A multiagent system for controlling building environments. In *P*roceedings of the First International Conference on Multiagent Systems, Menlo Park, Calif., pages 171–176, 1995.

[HKK97] E.H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *ACM SIGMOD*, 1997.

[HKM+88] J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham, and M. West. Scale and performance in a distributed file system. *A*CM Transactions on Computer Systems, 6(1):51–81, Feb 1988.

[HO01] John H. Hartman and John K. Ousterhout. The Zebra striped network file system. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *H*igh Performance Mass Storage and Parallel I/O: Technologies and Applications, pages 309–329. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[Hol59] J. Holland. A universal computer capable of executing an arbitrary number of subprograms simultaneously. In *P*roceedings of the 1959 Eastern Joint Computer Conference, pages 108–113, 1959.

[Hol00] K. Holtman. Object level physics data replication in the Grid. In *P*roceedings of ACAT'2000, pages 244–246, October 2000.

[HR95] B. Hayes-Roth. An architecture for adaptive intelligent systems. *A*rtificial Intelligence, 72(1-2):329–365, 1995.

[HRC95] M. Harry, J. Rosario, and A. Choudhary. VIP-FS: A virtual, parallel file system for high performance parallel and distributed computing. In *P*roceedings of the Ninth International Parallel Processing Symposium, April 1995.

[HS95] H. Haugeneder and D. Steiner. Cooperating agents: Concepts and applications. In *P*roceedings of the Agent Software Seminar. London, England. Unicom Seminars Ltd, pages 80–106, 1995.

[HS96] A. Haddadi and K. Sundermeyer. *B*elief-Desire-Intention Agent Architectures, pages 169–186. 1996. Article belonging to [OJ96].

[Hug02] Gordon F. Hugues. Wise drives. *I*EEE Spectrum, Aug 2002.

[HX98] K. Hwang and Z. Xu. *S*calable Parallel Computing: Technology, Architecture, Programming. WCB/McGraw-Hill, 1998.

[HYGO94] S. Hägg, F. Ygge, R. Gustavsson, and H. Ottosson. DA-Soc: A testbed for modelling distributed automation applications using agent-oriented programming. In *P*roceedings of the Sixth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-94), pages 39–51, 1994.

[Häg97] Staffan Hägg. Agent technology in industrial applications. In *P*roceedings of the Australia-Pacific Forum on Intelligent Processing and Manufacturing of Materials (IPMM'97), 1997.

[Inm84] INMOS Ltd. *OCCAM*: Programming Manual, 1984.

[Inm86] INMOS Ltd. *T*ransputer Reference Manual, 1986.

[Jal94] Pankaj Jalote. *F*ault tolerance in distributed systems. Prentice Hall, April 1994.

[JCB02] Hai Jin, Toni Cortes, and Rajkumar Buyya, editors. *H*igh Performance Mass Storage and Parallel I/O. IEEE Press, 2002.

[JCL95a] N. R. Jennings, J. M. Corera, and L. Laresgoiti. Developing industrial multiagent systems. In *P*roceedings of the First International Conference on Multiagent Systems, pages 423–430, 1995.

[JCL+95b] N. R. Jennings, J. M. Corera, L. Laresgoiti, E. H. Mamdani, F. Perriollat, P. Skarek, and L. Z. Varga. Using ARCHON to develop real-world DAI applications for electricity transportation management and particle accelerator control. *IEEE* Expert, 1995.

[Jen92] N. R. Jennings. Using GRATE to build cooperating agents for industrial control. In *P*roceedings of the IFAC/IFIP/IMACS International Symposium on Artificial Intelligence in Real Time Control, pages 691–696, 1992.

[JER+95] James V. Huber Jr., Christopher L. Elford, Daniel A. Reed, Andrew A. Chien, and David S. Blumenthal. PPFS: A high performance portable parallel file system. In *P*roceedings of the 9th ACM International Conference on Supercomputing, pages 385–394, Jul 1995.

[JFL+01] N. R. Jennings, P. Faratin, A. R. Lomuscio, S. Parsons, C. Sierra, and M. Wooldridge. Automated negotiation: Prospects, methods and challenges. *I*nternational Journal of Group Decision and Negotiation, 10(2):199–215, 2001.

[JFN+96] N. R. Jennings, P. Faratin, T. J. Norman, P. O'Brien, M. E. Wiegand, C. Voudouris, J. L. Alty, T. Miah, and E. H. Mamdani. ADEPT: Managing business processes using intelligent agents. In *P*roceedings of the BCS Expert Systems 96 Conference, Cambridge, UK, pages 5–23, 1996.

[JGN99] W. Johnston, D. Gannon, and W. Nitzberg. Grids as production computing environments: The engineering aspects of NASA's information power Grid. In *P*roc. 8th IEEE Symp. on High Performance Distributed Computing, 1999.

[JHD90] A. Tate J. Hendler and M. Drummond. AI planning: Systems and techniques. *A*I Magazine, 11(2):61–77, 1990.

[JML+92] N. R. Jennings, E. H. Mamdani, I. Laresgoiti, J. Pérez, and J. Corera. GRATE: A general framework for cooperative problem solving. *I*EE-BCS Journal of Intelligent Systems Engineering, 1(2):102–114, 1992.

[JSW98] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems Journal*, 1(1):7–38, 1998.

[JW98] Nicholas R. Jennings and Michael J. Wooldridge, editors. *A*gent Technology: Foundations, Applications and Markets. UNICOM-Seminars. Springer-Verlag, 1998.

[Kae91] L. P. Kaelbling. A situated automata approach to the design of embedded agents. *S*IGART Bulletin, 2(4):85–88, 1991.

[Kan01] Mahmut Kandemir. Compiler-directed collective I/O. *I*EEE Transactions on Parallel and Distributed Systems, 12(12):1318–1331, December 2001.

[Kaz88] M. L. Kazar. Synchronization and caching issues in the Andrew file system. In *P*roceedings of the Summer USENIX Technical Conference, February 1988.

[KCF+96] T. Kimbrel, P. Cao, E. Felten, A. Karlin, and K. Li. Integrated parallel prefetching and caching, 1996.

[KE91] David Kotz and Carla Schlatter Ellis. Practical prefetching techniques for parallel file systems. In *P*roceedings of the First International Conference on Parallel and Distributed Information Systems, pages 182–189. IEEE Computer Society Press, 1991.

[KGP89] R.H. Katz, G. Gibson, and D. Patterson. Disk system architectures for high performance computing. In *P*roceedings of the IEEE, volume 77(12), pages 1842–1858, December 1989.

[KH81] William A. Kornfeld and Carl E. Hewitt. The scientific community metaphor. *I*EEE Trans. on Systems, Man, and Cybernetics, pages 24–33, January 1981.

[KJ98] Michael Knapik and Jay Johnson. *D*eveloping Intelligent Agents for Distributed Systems. Computing McGraw-Hill, 1998.

[KLR+92] D. Kinny, M. Ljungberg, A. S. Rao, E. Sonenberg, G. Tidhar, and E. Werner. Planned team activity. In C. Castelfranchi and E. Werner, editors, *A*rtificial Social Systems - Selected Papers from the Fourth European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-92)(LNAI Volume 830), pages 226–256. Springer-Verlag: Heidelberg, Germany, 1992.

[KN94] D. Kotz and N. Nieuwejaar. Dynamic file-access characteristics of a production parallel scientific workload. In *P*roceedings of Supercomputing '94, pages 640–649, November 1994.

[KN95] D. Kotz and N. Nieuwejaar. File system workload on a scientific multiprocessor. *I*EEE Parallel and Distributed Technology. Systems and Applications, pages 134–154, 1995.

[Kna96] Frederick C. Knabe. An overview of mobile agent programming. In *P*roceedings of the 5th LOMAPS Workshop on Analysis and Verification of Multiple-Agent Languages, June 1996.

[Kor90] K. Korner. Intelligent caching for remote file service. In *P*roceedings of the 10th Intl. Conf. on Distributed Computing Systems, pages 220–226, May 1990.

[Kot93] David Kotz. Multiprocessor file system interfaces. In *P*roceedings of the Second International Conference on Parallel and Distributed Information Systems, pages 194–201, 1993.

[Kot94] David Kotz. Disk-directed I/O for MIMD multiprocessors. In *P*roceedings of the 1994 Symposium on Operating Systems Design and Implementation, pages 61–74, November 1994.

[KR90] L. P. Kaelbling and S. J. Rosenschein. Action and planning in embedded agents. *D*esigning Autonomous Agents, pages 35–48, 1990.

[Kri94] O. Krieger. *HFS*: A Flexible File System for Shared-Memory Multiprocessors. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 1994.

[KV94] P. Krishnan and J. S. Vitter. Optimal prediction for prefetching in the worst case. In *SODA*: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), 1994.

[Lab96] Y. Labrou. *S*emantics for an agent communication language. PhD thesis, Computer Science and Electrical Engineering Department (CSEE). University of Maryland, 1996.

[LAMa] LAM / MPI parallel computing MPI implementation list. *http://www.lam-mpi.org/mpi/implementations*.

[LAMb] LAM / MPI parallel computing. *http://www.lam-mpi.org*.

[Lam83] B. W. Lampson. Hints for computer system design. In *P*roceedings of the 9th ACM Symposium on Operating Systems Principles (SOSP), volume 17, pages 33–48, 1983.

[Law81] F. D. Lawlor. Efficient mass storage parity recovery mechanism. *IBM Technical Disclosure Bulletin*, 24(2):986–987, July 1981.

[LCD87] V. R. Lesser, D. D. Corkill, and E. H. Durfee. An update on the distributed vehicle monitoring testbed. Technical Report 87-111, University of Massachusetts, 1987.

[LE80] V. R. Lesser and L. D. Erman. Distributed interpretation: A model and experiment. *I*EEE Transaction on Computers, 29(12), 1980.

[LEG+97] P.M. Lyster, K. Ekers, J. Guo, M. Harber, D. Lamich, J.W. Larson, R. Lucchesi, R. Rood, S. Schubert, W. Sawyer, M. Sienkiewicz, A. da Silva, J. Stobie, L.L. Takacs, R. Todling, and J. Zero. Parallel computing at the NASA data assimilation office (DAO). San Jose, CA, November 1997. IEEE Computer Society Press.

[Les89] Y. Lespérance. A formal account of self knowledge and action. In *P*roceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89). Detroit, MI, pages 868–874, 1989.

[Les91] Victor R. Lesser. A retrospective view of FA/C distributed problem solving. *I*EEE Transactions on Systems, Man, and Cybernetics, 21(6):1347–1362, - 1991.

[LF97a] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Baltimore, MD 21250, 1997.

[LF97b] Yannis Labrou and Tim Finin. A proposal for a new KQML specification. Technical Report TR CS–97–03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, February 1997.

[LIN+93] Susan J. LoVerso, Marshall Isman, Andy Nanopoulos, William Nesheim, Ewan D. Milne, and Richard Wheeler. SFS: A parallel file system for the CM-5. In *P*roceedings of the 1993 Summer Usenix Conference, pages 291–305, 1993.

[LK91] E. Lee and R. Katz. Performance consequences of parity placement in disk arrays. In *P*roceedings of the ACM International Conference on Architectural Support for Programming Languages and Operating Systems, pages 190–199, 1991.

[LL92] M. Ljunberg and A. Lucas. The OASIS traffic management system. In *P*roceedings of the Second Pacific Rim International Conference on AI (PRICAI-92), Seoul, Korea, Sep 1992.

[LL93] S. Labidi and W. Lejouad. De lìntelligence artificielle distribuée aux systèmes multi-agents. Technical Report 2004, INRIA Institut National de Recherche en Informatique et en Automatique, 1993.

[LLC91] S. Lander, V. R. Lesser, and M. E. Connell. Conflict-resolution strategies for cooperating expert agents. In S. M. Deen, editor, *P*roceedings of the International Working Conference on Cooperating Knowledge-Based Systems (CKBS-90), pages 183–200. Springer-Verlag: New York, 1991.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor: A hunter of idle workstations. In *P*roc. of the 8th International Conference of Distributed Computing Systems, pages 104–111, June 1988.

[LO98] D. B. Lange and M. Oshima. *P*rogramming and Deploying Java Mobile Agents With Aglets. Addison-Wesley Pub Co, August 1998.

[LS90] Eliezer Levy and Abraham Silberschatz. Distributed file systems: concepts and examples. *ACM Computing Surveys*, 22(4):321–374, December 1990.

[LS93] C. M. Lewis and K. Sycara. Reaching informed agreement in multispecialist cooperation. *G*roup Decision and Negotiation, 2(3):279–300, 1993.

[LT96] Edward K. Lee and Chandramohan A. Thekkath. Petal: Distributed virtual disks. In 7th Intl. Conf. on Architectural Support for Programs, Languages and Operating Systems, pages 84–96, October 1996.

[Lu99] Hongjiu Lu. NFS server in linux: Past, present and future. *f*tp://ftp.valinux.com/pub/support/hjl/, August 1999.

[Mae89] P. Maes. The dynamics of action selection. In *P*roceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89). Detroit, MI, pages 991–997, 1989.

[Mae90a] P. Maes. Situated agents can have goals. *D*esigning Autonomous Agents, pages 49–70, 1990.

[Mae90b] Pattie Maes, editor. *D*esigning Autonomous Agents. MIT Press, Cambridge, 1990.

[Mae91] P. Maes. The agent network architecture (ANA). *S*IGART Bulletin, 2(4):115–120, 1991.

[MC95] F. G. McCabe and K. L. Clark. *A*pril - agent process interaction language, pages 324–340. Jan 1995. Article belonging to [WJ95b].

[MCd96] B. Moulin and B. Chaib-draa. *A*n Overview of Distributed Artificial Intelligence, pages 3–56. 1996. Chapter belonging to [OJ96].

[Mic96] Microsoft Corporation. *DCOM* Technical Overview. White Paper, 1996.

[MJLF84] Marshall K. McKusick, WilliamÑ. Joy, Samuel J. Leffler, and Robert S. Fabry. A fast file system for UNIX. *C*omputer Systems, 2(3):181–197, 1984.

[MK91a] L.W. McVoy and S. R. Kleiman. Extent-like performance from a UNIX file system. In *P*roceedings of the 1991 Winter USENIX Technical Conference (Dallas, TX), January 1991.

[MK91b] Ethan L. Miller and Randy H. Katz. Input/output behavior of supercomputer applications. In *P*roceedings of Supercomputing '91, pages 567–576, November 1991.

[MKKS01] Mihhail Matskin, Ole Jorgen Kirkeluten, Svenn Bjarte Krossnes, and Oystein Sale. Agora: An infrastructure for cooperative work support in multi-agent systems. *I*nfrastructure for Agents, Multi-Agent Systems, and Scalable Multi-Agent Systems, pages 28–40, March 2001.

[ML91] M. W. Mutka and M. Livny. The available capacity of a privately owned workstation environment. *P*erformance Evaluation, 12(4):269–284, 1991.

[Moo65] Gordon E. Moore. Cramming more components onto integrated circuits. *E*lectronics, 38(8), Apr 1965.

[Moo75] Gordon E. Moore. Progress in digital integrated electronics. In *I*EEE Digital Integrated Electronic Device Meeting, page 11, 1975.

[Moo90] R. C. Moore. *A* Formal Theory of Knowledge and Action, pages 480–519. 1990. Article belonging to [AHT90].

[Mor89] L. Morgenstern. Knowledge preconditions for actions and plans. In *P*roceedings of the Tenth International Joint Conference on Artificial Intelligence (IJCAI-87). Milan, Italy, pages 867–874, 1989.

[MP94] J. P. Müller and M. Pischel. Modelling interacting agents in dynamic environments. In *P*roceedings of the Eleventh European Conference on Artificial Intelligence (ECAI-94). Amsterdam, The Netherlands, pages 709–713, 1994.

[MPIa] MPICH-a portable implementation of MPI. *http://www-unix.mcs.anl.gov/mpi/mpich.*

[MPIb] MPI Forum. *http://www.mpi-forum.org/docs/docs.html.*

[MPIc] The Message Passing Interface (MPI) standard. *http://www-unix.mcs.anl.gov/mpi.*

[MPI94] Message Passing Interface Forum. *MPI*: A Message-Passing Interface Standard, May 1994.

[MPI96] MPI-IO: a parallel file I/O interface for MPI. The MPI-IO Committee, April 1996. Version 0.5.

[MPI97] Message Passing Interface Forum. *MPI*-2: Extensions to the Message-Passing Interface, Jul 1997.

[MPT94] J. P. Müller, M. Pischel, and M. Thiel. *A* pragmatic approach to modeling autonomous interacting systems, pages 226–240. 1994. Article belonging to [WJ94].

[MPT95] J. P. Müller, M. Pischel, and M. Thiel. *M*odelling reactive behavior in vertically layered agent architectures, pages 261–276. 1995. Article belonging to [WJ95b].

[MR97] Tara M. Madhyastha and Daniel A. Reed. Input/Output access pattern classification using hidden Markov models. In *P*roceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems, pages 57–67, San Jose, CA, 1997. ACM Press.

[MR00] M.R. Martinez and N. Roussopoulos. MOCHA: A self-extensible database middleware system for distributed data sources. In *P*roceedings of the ACM SIGMOD International Conference on Management of Data, Dallas, TX, May 2000.

[MS94] S. A. Moyer and V. S. Sunderam. PIOUS: A scalable parallel I/O system for distributed computing environments. In *P*roceedings of the Scalable High-Performance Computing Conference, pages 71–78, 1994.

[MS95] S. A. Moyer and V. S. Sunderam. Scalable concurrency control for parallel file systems. Technical Report CSTR-950202, Department of Math and Computer Science, Emory University, Atlanta, GA 30322, USA, 1995.

[Mue95] A. Mueller. Fast sequential and parallel algorithms for association rule mining: A comparison. Technical Report CS-TR-3515, University of Maryland, College Pak, 1995.

[MW96] T. Mullen and M. P. Wellman. Some issues in the design of market-oriented agents. In M. Wooldridge, J.-P. Müller, and M. Tambe, editors, *I*ntelligent Agents II — Agent Theories, Architectures, and Languages (LNAI 1037), pages 283–298. Springer-Verlag: Heidelberg, Germany, 1996.

[MZ00] William A. Maniatty and Mohammed J. Zaki. Systems supports for scalable data mining. In *S*IGKDD Explorations. ACM SIGKDD, volume 2, pages 56–65, December 2000.

[Mül94] J. P. Müller. A conceptual model for agent interaction. In S. M. Deen, editor, *P*roceedings of the Second International Working Conference on Cooperating Knowledge Based Systems (CKBS-94), pages 213–234, 1994.

[Mül96] J. P. Müller. *A*n Architecture for Dynamically Interacting Agents. PhD thesis, Germain AI Research Center (DFKI GmbH), Saarbrücken, 1996.

[Nec94] R.D. Neches. Overview of the ARPA knowledge sharing effort. Edited by Gruber, April 1994.

[New82] A. Newell. The knowledge level. *A*rtificial Intelligence, 18:87–127, 1982.

[NF96] J. Nieplocha and I. Foster. Disk resident arrays: An array-oriented I/O library for out-of-core computations. In *P*roceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, pages 196–204, October 1996.

[NFK98] Jarek Nieplocha, Ian Foster, and Rick A. Kendall. ChemIO: High-performance parallel I/O for computational chemistry applications. *I*nt. J. Supercomp. Apps. High Perf. Comp., 12(3), 1998.

[NFS89] Network Working Group. *NFS*: Network File System Protocol Specification, March 1989. RFC 1094.

[NFS91] Raymond T. Ng, Christos Faloutsos, and Timos K. Sellis. Flexible buffer allocation based on marginal gains. In James Clifford and Roger King, editors, *P*roceedings of the 1991 ACM SIGMOD International Conference on Management of Data, Denver, Colorado, May 29-31, 1991, pages 387–396. ACM Press, 1991.

[NFS95] Network Working Group. *NFS* Version 3 Protocol Specification, June 1995. RFC 1813.

[NIS02] Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, , CA, 95054, U.S.A. *S*ystem Administration Guide: Naming and Directory Services (DNS, NIS, and LDAP), 2002.

[Nit92] Bill Nitzberg. Performance of the iPSC/860 concurrent file system. Technical Report Technical Report RND-92-020, NAS Systems Division, NASA Ames, December 1992.

[NK96] Nils Nieuwejaar and David Kotz. The Galley parallel file system. In *P*roceedings of the 10th ACM International Conference on Supercomputing, pages 374–381. ACM Press, May 1996.

[NKP+95] N. Nieuwejaar, D. Kotz, A. Purakayastha, C.S. Ellis, and M. Best. File-access characteristics of parallel scientific workloads. Technical Report PCS-TR95-263, Darmouth College Computer Science, 1995.

[NL91] B. Nitzberg and V. Lo. Distributed shared memory: A survey of issues and algorithms. *I*EEE Computer, 24(8):52–60, 1991.

[NS76] A. J. Newell and H. A. Simon. Computer science as empirical enquiry. *C*ommunications of the ACM, 19:113–126, 1976.

[NsPCC02] Jaechun No, Sung soon Park, Jesús Carretero, and AlokÑ. Choudhary. Design and implementation of a parallel i/o runtime system for irregular applications. *J*ournal of Parallel and Distributed Computing, 62(2):193–220, 2002.

[NST99] NSTL. Defragmentation performance testing. Technical Report NSTL Final Report, NSTL, November 1999.

[NWO88] MichaelÑ. Nelson, Brent B. Welch, and John K. Ousterhout. Caching in the Sprite network file system. *A*CM Transactions on Computer Systems, 6(1):134–154, Feb 1988.

[OCD+89] John K. Ousterhout, Andrew R. Cherenson, Frederick Douglis, MichaelÑ. Nelson, and Brent B. Welch. The Sprite network operating system. *I*EEE Computer, 21(2):23–35, February 1989.

[OCH+85] John Ousterhout, Hervé Da Costa, David Harrison, John Kunzeand Mike Kupfer, and James Thompson. A trace driven analysis of the UNIX 4.2 BSD file system. In *P*roceedings of the Tenth ACM Symposium on Operating Systems Principles, pages 15–24, December 1985.

[OJ96] G. M. O'Hare and N. R. Jennings, editors. *F*oundations of Distributed Artificial Intelligence. John Wiley and Sons, 1996.

[OMG01] Object Management Group. *C*omplete CORBA 2.4.2 Specification. Document formal/01-02-33. http://www.omg.org, 2001.

[Par87] V. Parunak. Manufacturing experience with the Contract Net. *D*istributed Artificial Intelligence, Volume I, pages 285–310, 1987.

[Pat94] Y.Ñ. Patt. The I/O subsystem: A candidate for improvement. *I*EEE Computer, 27(3):15–16, March 1994.

[PB86] A. Park and K. Balasubramanian. Providing fault tolerance in parallel secondary storage systems. Technical Report CS-TR-057-86, Department of Computer Science, Princeton University, November 1986.

[PBB+99] Kenneth W. Preslan, Andrew P. Barry, Jonathan E. Brassow, Grant M. Erickson, Erling Nygaard, Christopher J. Sabol, Steven R. Soltis, David C. Teigland, and Matthew T. O'Keefe. A 64-bit, shared disk file system for linux. *h*ttp://www.sistina.com/gfs/Pages/gfspapers.html, March 1999.

[PCG+03a] María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. A flexible multiagent parallel file system for clusters. *I*nternational Workshop on Parallel I/O Management Techniques (PIOMT'2003) (Lecture Notes in Computer Science), June 2003.

[PCG+03b] María S. Pérez, Jesús Carretero, Félix García, José M. Peña, and Víctor Robles. MAPFS: A flexible infrastructure for data-intensive grid applications. In *1*st European Across Grids Conference, February 2003.

[PCGK89] D. A. Patterson, P. M. Chen, G. A. Gibson, and R. H. Katz. Introduction to Redundant Arrays of Inexpensive Disks (RAID). In *P*roceedings of IEEE COMPCON, pages 112–117, 1989.

[PCY95] J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In *ACM SIGMOD*, 1995.

[PEK+94] Apratim Purakayastha, Carla Schlatter Ellis, David Kotz, Nils Nieuwejaar, and Michael Best. Characterizing parallel file-access patterns on a large-scale multiprocessor. Technical Report Technical Report CS-1994-33, Dept. of Computer Science, Duke University, October 1994.

[PGC01] María S. Pérez, Félix García, and Jesús Carretero. A new multiagent based architecture for high performance I/O in clusters. *2*001 International Conference on Parallel Processing Workshops, September 2001.

[PGC02] María S. Pérez, Félix García, and Jesús Carretero. MAPFS_MAS: A model of interaction among information retrieval agents. In *2*nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), May 2002.

[PGG+01] R. Hugo Patterson, Garth A. Gibson, Eka Ginting, Daniel Stodolsky, and Jim Zelenka. Informed prefetching and caching. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *H*igh Performance Mass Storage and Parallel I/O: Technologies and Applications, pages 224–244. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[PGK88] D. A. Patterson, G. Gibson, and R. H. Katz. A case for Redundant Arrays of Inexpensive Disks (RAID). In *P*roceedings of ACM SIGMOD, pages 109–116, June 1988.

[PGS93] R. Hugo Patterson, Garth A. Gibson, and M. Satyanarayanan. A status report on research in transparent informed prefetching. *A*CM Operating Systems Review, 27(2):21–34, 1993.

[Pie89] P. Pierce. A concurrent file system for a highly parallel mass storage subsystem. In *P*roceedings of the Fourth Conference on Hypercubes Concurrent Computers and Applications HCCA, pages 155–161, Mar 1989.

[PLS92] Mike P. Papazoglou, Steven C. Lanfmanu, and Timos K. Sellis. An organizational framework for cooperating intelligent information systems. *I*nternational Journal of Intelligent and Cooperative Information Systems, 1(1):169–202, 1992.

[Pog95] A. Poggi. *D*AISY: an Object-Oriented System for Distributed Artificial Intelligence, pages 341–354. Jan 1995. Article belonging to [WJ95b].

[PP93] Barbara K. Pasquale and George C. Polyzos. A static analysis of I/O characteristics of scientific applications in a production workload. In *P*roceedings of Supercomputing '93, pages 388–397, 1993.

[PPG+02a] María S. Pérez, Ramón A. Pons, Félix García, Jesús Carretero, and María L. Córdoba. An optimization of Apriori algorithm through the usage of parallel I/O and hints. *R*ough Sets and Current Trends in Computing (LNAI 2475), October 2002.

[PPG+02b] María S. Pérez, Ramón A. Pons, Félix García, Jesús Carretero, and Víctor Robles. A proposal for I/O access profiles in parallel data mining algorithms. In *3*rd ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, June 2002.

[PPT+01] A. Pascual, O.M. Pérez, O. Trelles, R. Pascual-Marqui, and J.M. Carazo. Nueves redes neuronales auto-organizativas para el análisis y agrupamiento de patrones de expresión génica. In *I*I Jornadas de Bioinformática. Málaga, July 2001.

[PSV94] Vinay Sadananda Pai, Alejandro A. Schäffer, and Peter J. Varman. Markov analysis of multiple-disk prefetching strategies for external merging. *T*heoretical Computer Science, 128(1–2):211–239, 1994.

[Rao96] Anand S. Rao. *D*ecision Procedures for Propositional Linear-Time Belief-Desire-Intention logics, pages 33–48. 1996. Article belonging to [WMT96].

[RAPL+00] A. Rodriguez, A.Pérez-Pulido, A.D. López, G. Thode, JM. Carazo, and O. Trelles. Mining low-level similarity signals from sequence databases. In *T*he 4th World Multiconference on Systemics, Cybernetics and Informatics SCI. Orlando, 2000.

[RB89] J. S. Rosenschein and J. S. Breese. *C*ommunication-free interactions among rational agents: A probabilistic approach, pages 99–118. 1989. Article belonging to [GH89].

[RBK92] K.K. Ramakrishnan, P. Biswas, and R. Karedla. Analysis of file I/O traces in commercial computing environments. In *P*roceedings of the 1992 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Newport, Rhode Island, pages 78–90, June 1992.

[RG88] J. S. Rosenschein and M. R. Genesereth. *D*eals among rational agents, pages 227–234. Springer-Verlag, 1988. Article belonging to [BG88c].

[RG91a] Anand S. Rao and Michael P. Georgeff. Asymmetry thesis and side-effect problems in linear time and branching time intention logics. In *P*roceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91), Sydney, Australia, pages 498–504, 1991.

[RG91b] Anand S. Rao and Michael P. Georgeff. Modeling rational agents within a BDI-architecture. In James Allen, Richard Fikes, and Erik Sandewall, editors, *P*roceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA, 1991.

[RG92a] Anand S. Rao and Michael P. Georgeff. An abstract architecture for rational agents. In C. Rich, W. Swartout, and B. Nebel, editors, *P*roceedings of knowledge Representation and Reasoning 92, pages 439–449, 1992.

[RG92b] Anand S. Rao and Michael P. Georgeff. Social plans: Preliminary report. In E. Werner and Y. Demazeau, editors, *D*ecentralized AI 3 - Proceedings of the Third European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-91), pages 57–76. Elsevier Science Publisher B.V: Amsterdam, The Netherlands, 1992.

[RG93] Anand S. Rao and Michael P. Georgeff. A model-theoretic approach to the verification of situated reasoning systems. In *P*roceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), Chambéry, France, pages 318–324, 1993.

[RG95] Anand S. Rao and Michael P. Georgeff. BDI agents: From theory to practice. In *P*roceedings of the First International Conference on Multi-Agent Systems (ICMAS-95). San Francisco. USA., June 1995.

[RK86] S. Rosenschein and L. P. Kaelbling. The synthesis of digital machines with provable epistemic properties. In J. Y. Halpern, editor, *P*roceedings of the 1986 Conference on Theoretical Aspects of Reasoning About Knowledge, pages 83–98. Morgan Kaufmann Publishers: San Mateo, CA, 1986.

[RK94] E. Rich and K. Knight. *I*nteligencia Artificial. McGraw-Hill, 1994.

[RMA+96] Daniel A. Reed, Tara M. Madhyastha, Ruth A. Aydt, Christopher L. Elford, Will H. Scullin, and Evgenia Smirni. I/O, performance analysis, and performance data immersion. In *P*roceedings of MASCOTS'96, pages 5–15, 1996.

[RN94] S. Russell and P. Norvig. *A*rtificial Intelligence: A Modern Approach. Englewood Cliffs, NJ. Prentice-Hall, 1994.

[RO92] M. Rosenblum and J. Ousterhout. The design and implementation of a Log-Structured File System. *A*CM Transactions on Computer Systems, 10(1):26–52, February 1992.

[Ros85] S. Rosenschein. Formal theories of knowledge in AI and robotics. *N*ew Generation Computing, pages 345–357, 1985.

[RP95] Brad Rullman and David Payne. An efficient file I/O interface for parallel applications. In *D*RAFT presented at the Workshop on Scalable I/O, Frontiers '95, February 1995.

[RTL96] W. Gropp Rajeev Thakur and E. Lusk. An abstract-device interface for implementing portable parallel-I/O interfaces. In *P*roceedings of the 6th Symposium on the Frontiers of Massively Parallel Computation, pages 180–187, October 1996.

[SA91] M. P. Singh and N. M. Asher. Towards a formal theory of intentions. In *L*ogics in AI - Proceedings of the European Workshop JELIA-90 (LNAI Volume 478), pages 472–486. Springer-Verlag: Heidelberg, Germany, 1991.

[SACR01] Evgenia Smirni, Ruth A. Aydt, Andrew A. Chien, and Daniel A. Reed. I/O requirements of scientific applications: An evolutionary view. In Hai Jin, Toni Cortes, and Rajkumar Buyya, editors, *H*igh Performance Mass Storage and Parallel I/O: Technologies and Applications, pages 576–594. IEEE Computer Society Press and Wiley, New York, NY, 2001.

[San93] T. W. Sandholm. An implementation of the Contract Net Protocol based on marginal cost calculations. In *P*roceedings of the 12th International Workshop on Distributed Artificial Intelligence, pages 295–308, Hidden Valley, Pennsylvania, 1993.

[SAvL96] L. Sommaruga, N. M. Avouris, and M. H. van Liedekerke. *T*he evolution of the Coopera platform, pages 365–400. 1996. Article belonging to [OJ96].

[SB99] Luís Moura E Silva and Rajkumar Buyya. *P*arallel Programming Models and Paradigms, chapter Programming Environments and Development Tools. Prentice Hall, 1999. Chapter belonging to [Buy99b].

[SBN82] D. Siewiorek, C. Bell, and A. Newell. *C*omputer Structures: Principles and Examples. MacGraw-Hill, Pittsbutgh, Pennsylvania, 1982.

[SCHR+88] R. Steeb, S. Cammarata, F. A. Hayes-Roth, P. W. Thorndyke, and R. B. Wesson. *D*istributed intelligence for air fleet control, pages 102–105. Springer-Verlag, 1988. Article belonging to [BG88c].

[SCJ+95] K. Seamons, Y. Chen, P. Jones, J. Jozwiak, and M. Winslett. Server-directed collective I/O in Panda. In *P*roceedings of Supercomputing '95, San Diego, CA. IEEE Computer Society Press, December 1995.

[SD81] R. G. Smith and R. Davis. Frameworks for cooperation in distributed problem solving. *S*ystems, Man and Cybernetics, 11(1):61–69, January 1981.

[SD83] R. Smith and R. Davis. Negotiation as a metaphor for distributed problem solving. *A*rtificial Intelligence, 20:63–109, 1983.

[Sea96] K. Seamons. *P*anda: Fast Access to Persistent Arrays Using High Level Interfaces and Server Directed Input/Output. PhD thesis, University of Illinois at Urbana-Champaign, 1996.

[SF89] A. Sathi and M. Fox. *C*onstraint-directed Negotiation of Resource Reallocation, pages 163–195. 1989. Article belonging to [GH89].

[SGK+85] R. Sandberg, D. Goldberg, S. Kleiman, D Walsh, and B. Lyon. Design and implementation of the SUN network filesystem. In *P*roceedings of the 1985 USENIX Conference, 1985.

[Sho91] Y. Shoham. AGENTO: A simple agent language and its interpreter. In *P*roceedings of the Ninth National Conference on Artificial Intelligence (AAAI-91). Anaheim CA, pages 704–709, 1991.

[Sho93] Yoav Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[Sho97] Yoav Shoham. An overview of agent-oriented programming. *Software Agents*, 1997.

[Sin90] M. P. Singh. Towards a theory of situated know-how. In *P*roceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90). Stockholm, Sweden, pages 604–609, 1990.

[Sin91a] M. P. Singh. Group ability and structure. In Y. Demazeau and J. P. Müller, editors, *D*ecentralized AI 2 - Proceedings of the Second European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-90), pages 127–146. Elsevier Science Publisher B. V.: Amsterdam, The Netherlands, 1991.

[Sin91b] M. P. Singh. Towards a formal theory of communication for multi-agent systems. In *P*roceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91). Sydney, Australia, pages 69–74, 1991.

[Sin92] M. P. Singh. A critical examination of the Cohen-Levesque theory of intention. In *P*roceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92). Vienna, Austria, pages 364–368, 1992.

[Sin94] M. P. Singh. Multiagent systems: A theoretical framework for intentions, know-how and communications (LNAI volume 799). *Springer-Verlag*, 1994.

[SK96] T. Shintani and M. Kitsuregawa. Hash based parallel algorithms for mining association rules. In *4*th Int. Conf. Parallel and Distributed Information Systems, 1996.

[SK99] T. Shintani and M. Kitsuregawa. Parallel algorithms for mining association rule mining on large scale PC cluster. In Mohammed J. Zaki and Ching-Tien Ho, editors, *W*orkshop on Large-Scale Parallel KDD Systems, San Diego, CA, USA, August 1999. ACM. in conjunction with ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD99).

[SKK+90] M. Satyanarayanan, James J. Kistler, Puneet Kumar, Maria E. Okasaki, Ellen H. Siegel, and David C. Steere. Coda: A highly available file system for a distributed workstation environment. *I*EEE Transactions on Computers, 39(4):447–459, 1990.

[SMD94] J. Bresina S. Minton and M. Drummond. Total-order and partial-order planning: A comparative analysis. *J*ournal of Artificial Intelligence Research, 2:227–262, December 1994.

[Smi80] R. G. Smith. A framework for distributed problem solving. *U*MI Research Press, 1980.

[Smi81] A. J. Smith. Analysis of long term file reference patterns for application to file migration algorithms. *I*EEE Transactions on Software Engineering, 7(4):403–410, July 1981.

[Smi85] A. J. Smith. Disk cache miss ratio analysis and design considerations. *A*CM Trasactios on Computer Systems, 3(3):161–203, August 1985.

[Smi88] R. G. Smith. *T*he contract net protocol: High-level communication and control in a distributed problem solver, pages 357–366. Springer-Verlag, 1988. Article belonging to [BG88c].

[Sol97] Steven R. Soltis. *T*he Design and Implementation of a Distributed File System based on Shared Network Storage. PhD thesis, University of Minnesota, 1997.

[SPvVG96] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. Distributed intelligent agents. *I*EEE Expert, 11(6):36–46, December 1996.

[SPvVG01] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon, 2001.

[SQ93] U. M. Schwuttke and A. G. Quan. Enhancing performance of cooperating agents in real-time diagnostic systems. In *P*roceedings of the Thirteenth International Joint Conference on Artificial Intelligence (IJCAI-93), Menlo Park, Calif., pages 332–337, 1993.

[SR97] E. Smirni and D. A. Reed. Workload characterization of input/output intensive parallel applications. In *P*roceedings of the Conference on Modelling Techniques and Tools for Computer Performance Evaluation, volume 1245, pages 169–180. Springer-Verlag, 1997.

[Sri95a] R. Srinivasan. RPC: Remote procedure call. protocol specification version 2. *RFC 1831*, August 1995.

[Sri95b] R. Srinivasan. XDR: External data representation standard. *RFC 1832*, August 1995.

[SRO96] Steven R. Soltis, Thomas M. Ruwart, and Matthew T. O'Keefe. The Global File System. In *P*roceedings of the Fifth NASA Goddard Space Flight Center Conference on Mass Storage Systems and Technologies, September 1996.

[STE] Intelligent agents software classification. *http://w3.informatik.gu.se/ dixi/agent/class.htm.*

[Ste90] L. Steels. Cooperation between distributed agents through self organization. In Y. Demazeau and J. P. Müller, editors, *D*ecentralized AI - Proceedings of the First European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-89), pages 175–196. Elsevier Science Publisher B.V: Amsterdam, The Netherlands, 1990.

[Ste96] D. D. Steiner. *I*MAGINE: An integrated environment for constructing distributed artificial intelligence systems, pages 345–364. 1996. Article belonging to [OJ96].

[Ste98] W. Richard Stevens. *UNIX* Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI. Prentice Hall, 1998.

[Ste99] W. Richard Stevens. *UNIX* Network Programming, Volume 2, Second Edition: Interprocess Communications. Prentice Hall, 1999.

[Sto98] Heinz Stockinger. Dictionary on parallel Input/Output. *Master's Thesis*, February 1998.

[Sun01] Sun Microsystems, Inc. *E*nterprise JavaBeans Specification 2.0, Final Release, August 2001.

[Sun02] Sun Microsystems, Inc. *J*ava Remote Method Invocation Specification, 2002.

[Svo84] L. Svobodova. File servers for network-based distributed systems. *ACM Computing Surveys*, 16(4):353–398, December 1984.

[SW94] K. Seamons and M. Winslett. An efficient abstract interface for multidimensional array I/O. In *P*roceedings of Supercomputing '94, pages 650–659, November 1994.

[SW96] K. E. Seamons and M. Winslett. Multidimensional array I/O in Panda 1.0. *J*ournal of Supercomputing, 10(2):191–211, 1996.

[SWDC97] R. Stevens, P. Woodward, T. DeFanti, and C. Catlett. From the I-WAY to the national technology Grid. *C*ommunications of the ACM, 40(11):50–61, 1997.

[Syc89] K. Sycara. *M*ulti-agent compromise via negotiation, pages 119–139. 1989. Article belonging to [GH89].

[Syc97] K. Sycara. Using option pricing to value commitment flexibility in multi-agent systems. Technical Report CMU-CS-97-169, Carnegie Mellon University, 1997.

[Syc98] Katia P. Sycara. Multiagent systems. *A*I Magazine, 19(2), 1998.

[Tan87] A. S. Tanenbaum. *O*perating Systems: Design And Implementation. Prentice Hall, NJ-USA, 1987.

[Tan92] A. S. Tanenbaum. *M*odern Operating Systems. Prentice Hall, 1992.

[TBC+94] R. Thakur, R. Bordawekar, A. Choudhary, R. Ponnusamy, and T. Singh. PASSION runtime library for parallel I/O. In *P*roceedings of the Scalable Parallel Libraries Conference, pages 119–128, October 1994.

[TC02] Dave Turner and Xuehua Chen. Protocol-dependent message-passing performance on linux clusters. In *C*luster 2002 Conference, 2002.

[TD91] C. D. Tait and D. Duchamp. Detection and exploitation of file working sets. In *P*roceedings of the 11th International Conference on Distributed Computing Systems (ICDCS), pages 2–9, Washington, DC, 1991. IEEE Computer Society.

[TG96] S. Toledo and F. Gustavson. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations. In *P*roceedings of the 4th Annual Workshop on I/O in Parallel and Distributed Systems, pages 28–40, May 1996.

[TGL98] Rajeev Thakur, William Gropp, and Ewing Lusk. A case for using MPI's derived datatypes to improve I/O performance. In *S*ubmitted as an extended abstract to SC98: High Performance Networking and Computing. Argonne National Laboratory, May 1998.

[TGL99a] Rajeev Thakur, William Gropp, and Ewing Lusk. Data sieving and collective I/O in ROMIO. In *P*roceedings of the 7th Symposium on the Frontiers of Massively Parallel Computation, pages 182–189. Argonne National Laboratory, February 1999.

[TGL99b] Rajeev Thakur, William Gropp, and Ewing Lusk. On implementing MPI-IO portably and with high performance. In *P*roceedings of the Sixth Workshop on I/O in Parallel and Distributed Systems, pages 23–32, May 1999.

[TGL02] Rajeev Thakur, William Gropp, and Ewing Lusk. Optimizing noncontiguous accesses in MPI-IO. *P*arallel Computing, 28(1):83–105, 2002.

[Tho78] K. Thompson. UNIX implementation. *T*he Bell System Technical Journal, 6(2):1931–1946, 1978.

[Tho94] S. R. Thomas. *T*he PLACA Agent Programming Language, pages 307–319. 1994. Article belonging to [WJ94].

[TLJ+99] B. Tierney, J. Lee, W. Johnston, B. Crowley, and M. Holding. A network-aware distributed storage cache for data-intensive environments. In *P*roceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, pages 185–193, Redondo Beach, CA, Aug 1999.

[TML97] Chandramohan A. Thekkath, Timothy Mann, and Edward K. Lee. Frangipani: A scalable distributed file system. In *S*ymposium on Operating Systems Principles, pages 224–237, 1997.

[TOP99] http://www.top500.org, November 1999.

[Tri79] K. Trivedi. An analysis of prepaging. *C*omputing, 22(3), 1979.

[Try99] Mobile agents: The work force of the new millennium. *A* Tryllian Whitepaper, November 1999.

[TV91] M. Torrance and P. A. Viola. The AGENT0 manual. Technical report, Stanford University, 1991.

[VK96] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. *J*ournal of the ACM, 43(5):771–793, 1996.

[Voo94] E. M. Voorhees. Software agents for information retrieval. *1994 Spring Symposium*, pages 126–129, March 1994.

[VTF01] S. Vazhkuda, S. Tuecke, and I. Foster. Replica selection in the Globus data Grid. In *P*roceedings of the International workshop on Data Models and Databases on Clusters and the Grid (DataGrid2001). IEEE Computer Society Press, 2001.

[WC00] Steve Widen and Chris Chris. Disk defragmentation for windows NT/2000- hidden gold for the enterprise. Technical Report IDC white paper, IDC, 2000.

[Wer88a] E. Werner. Social intentions. In *P*roceedings of the European Conference on Artificial Intelligence (ECAI-88), pages 719–723, 1988.

[Wer88b] E. Werner. Toward a theory of communication and cooperation for multiagent planning. In M. Y. Vardi, editor, *P*roceedings of the Second Conference on Theoretical Aspects of Reasoning About Knowledge, pages 129–144. Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[Wer89] E. Werner. *C*ooperating agents: A unified theory of communication and social structure, pages 3–36. 1989. Article belonging to [GH89].

[Wer90] E. Werner. What can agents do together: A semantics of co-operative ability. In *P*roceedings of the Ninth European Conference on Artificial Intelligence (ECAI-90). Stockholm, Sweden, pages 694–701, 1990.

[Wer91] E. Werner. A unified view of information, intention and ability. In Y. Demazeau and J. P. Müller, editors, *D*ecentralized AI 2 - Proceedings of the Second European Workshop on Modelling Autonomous Agents and Multi-Agent Worlds (MAAMAW-90), pages 109–126. Elsevier Science Publisher B.V: Amsterdam, The Netherlands, 1991.

[WF92] M. Wooldridge and M. Fisher. A first-order branching time logic of multi-agent systems. In *P*roceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92). Vienna, Austria, pages 234–238, 1992.

[WG94] P. Wavish and M. Graham. *R*oles, skills and behaviour, pages 320–333. 1994. Article belonging to [WJ94].

[WGSS96] John Wilkes, Richard Golding, Carl Staelin, and Tim Sullivan. The HP AutoRAID hierarchical storage system. *ACM Transactions on Computer Systems*, 14(1):108–136, February 1996.

[Whi75] J. E. White. A high-level framework for network-based resource sharing. *RFC 707*, December 1975.

[Whi95] J. E. White. Telescript technology: Mobile agents. *G*eneral Magic White Paper, October 1995.

[WHRB+81] R. B. Wesson, F. A. Hayes-Roth, J. W. Burge, C. Stasz, and C. A. Sunshine. Network structures for distributed situation assesment. *I*EEE Transaction on Systems, Man and Cybernetics, 11(1), 1981.

[Wit92] T. Wittig, editor. *A*RCHON: an architecture for multi-agent systems. Ellis Horwood, 1992.

[WJ94] Michael J. Wooldridge and Nicholas R. Jennings, editors. *A*gent theories, architectures and languages. Springer-Verlag, 1994.

[WJ95a] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 1995.

[WJ95b] Michael J. Wooldridge and Nicholas R. Jennings, editors. *I*ntelligent Agents: Theories, Architectures and Languages (LNAI Volume 890). Springer-Verlag: Heidelberg, Germany, 1995.

[WL96] G. Wilson and P. Lu. *P*arallel Programming Using C++. MIT Press, Cambridge, MA, 1996.

[WMT96] M. Wooldridge, J. P. Müller, and M. Tambe, editors. *I*ntelligent Agents II. Springer-Verlag, 1996.

[Woo92] M. Wooldridge. *T*he Logical Modelling of Computational Multi-Agent Systems. PhD thesis, Department of Computation, UMIST, Manchester, UK, 1992.

[WP99] Tony White and Bernard Pagurek. Emergent behavior and mobile agents. 1999.

[WRR95] D. Weerasooriya, A. Rao, and K. Ramamohanarao. *D*esign of a Concurrent Agent-Oriented Language, pages 386–402. 1995. Article belonging to [WJ95b].

[WV94] R. Weihmayer and H. Velthuijsen. Application of distributed AI and cooperative problem solving to telecommunications. *A*I Approaches to Telecommunication and Network Management, 1994.

[WW97] H. Wang and C. Wang. Intelligent agents in the nuclear industry. *I*EEE Computer, 30(11):28–34, 1997.

[YFH+96] S.J. Young, G.Y. Fan, D. Hessler, S. Lamont, T.T. Elvins, M. Hadida, G. Hanyzewski, J.W. Durkin, P. Hubbard, G. Kindlmann, E. Wong, D. Greenberg, S. Karin, and M.H. Ellisman. Implementing a collaboratory for microscopic digital anatomy. *Supercomputer Applications and High Performance Computing*, 10(2/3):170–181, 1996.

[ZDL+89] L. Zahn, T. H. Dineen, P. J. Leach, E. A. Martin, N. W. Mishkin, J.Ñ. Pato, and G. L. Wyant. *N*etwork Computing Architecture. Prentice-Hall, 1989.

[ZOPL96] M. Zaki, M. Ogihara, S. Parthasarathy, and W. Li. Parallel data mining for association rules on shared-memory multi-processor machines. In *S*upercomputing'96, 1996.

[ZPOL97] M. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. Parallel algorithms for fast descovery of association rules. *D*ata Mining and Knowledge Discovery: An International Journal, 1(4):343–373, 1997.